```
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN  VVV         VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN  VVV         VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN  VVV         VVV
CCC              OOO         OOO  NNN         NNN  VVV         VVV
CCC              OOO         OOO  NNN         NNN  VVV         VVV
CCC              OOO         OOO  NNN         NNN  VVV         VVV
CCC              OOO         OOO  NNNNNN      NNN  VVV         VVV
CCC              OOO         OOO  NNNNNN      NNN  VVV         VVV
CCC              OOO         OOO  NNNNNN      NNN  VVV         VVV
CCC              OOO         OOO  NNN   NNN   NNN  VVV         VVV
CCC              OOO         OOO  NNN   NNN   NNN  VVV         VVV
CCC              OOO         OOO  NNN   NNN   NNN  VVV         VVV
CCC              OOO         OOO  NNN      NNNNNN  VVV         VVV
CCC              OOO         OOO  NNN      NNNNNN  VVV         VVV
CCC              OOO         OOO  NNN      NNNNNN  VVV         VVV
CCC              OOO         OOO  NNN         NNN     VVV   VVV
CCC              OOO         OOO  NNN         NNN     VVV   VVV
CCC              OOO         OOO  NNN         NNN     VVV   VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN        VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN        VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN        VVV
```

```
RRRRRRRR    EEEEEEEEEE    CCCCCCCC  LL          RRRRRRRR    EEEEEEEEEE    CCCCCCCC
RRRRRRRR    EEEEEEEEEE    CCCCCCCC  LL          RRRRRRRR    EEEEEEEEEE    CCCCCCCC
RR      RR  EE            CC        LL          RR      RR  EE            CC
RR      RR  EE            CC        LL          RR      RR  EE            CC
RR      RR  EE            CC        LL          RR      RR  EE            CC
RR      RR  EE            CC        LL          RR      RR  EE            CC
RRRRRRRR    EEEEEEEE      CC        LL          RRRRRRRR    EEEEEEEE      CC
RRRRRRRR    EEEEEEEE      CC        LL          RRRRRRRR    EEEEEEEE      CC
RR  RR      EE            CC        LL          RR  RR      EE            CC
RR  RR      EE            CC        LL          RR  RR      EE            CC
RR    RR    EE            CC        LL          RR    RR    EE            CC                ....
RR    RR    EE            CC        LL          RR    RR    EE            CC                ....
RR      RR  EEEEEEEEEE    CCCCCCCC  LLLLLLLLLL  RR      RR  EEEEEEEEEE    CCCCCCCC          ....
RR      RR  EEEEEEEEEE    CCCCCCCC  LLLLLLLLLL  RR      RR  EEEEEEEEEE    CCCCCCCC          ....

LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

```
;    1          0001  0 %TITLE  'VAX-11 CONVERT/RECLAIM'
;    2          0002  0 MODULE  RECL$REC        ( IDENT='V04-000',
;    3          0003  0                         OPTLEVEL=3
;    4          0004  0                         ) =
;    5          0005  0
;    6          0006  1 BEGIN
;    7          0007  1
;    8          0008  1 !**************************************************************************
;    9          0009  1 !*                                                                        *
;   10          0010  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                               *
;   11          0011  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                 *
;   12          0012  1 !*  ALL RIGHTS RESERVED.                                                   *
;   13          0013  1 !*                                                                        *
;   14          0014  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  *
;   15          0015  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;   16          0016  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;   17          0017  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  *
;   18          0018  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY  *
;   19          0019  1 !*  TRANSFERRED.                                                           *
;   20          0020  1 !*                                                                        *
;   21          0021  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE  *
;   22          0022  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT  *
;   23          0023  1 !*  CORPORATION.                                                           *
;   24          0024  1 !*                                                                        *
;   25          0025  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS  *
;   26          0026  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                *
;   27          0027  1 !*                                                                        *
;   28          0028  1 !*                                                                        *
;   29          0029  1 !**************************************************************************
```

```
;    31          0030  1 !++
;    32          0031  1 !
;    33          0032  1 ! Facility:     VAX-11 CONVERT/RECLAIM
;    34          0033  1 !
;    35          0034  1 ! Environment:
;    36          0035  1 !
;    37          0036  1 !               VAX/VMS Operating system
;    38          0037  1 !
;    39          0038  1 ! Abstract:
;    40          0039  1 !
;    41          0040  1 !               This module contains routines to handle index records.
;    42          0041  1 !
;    43          0042  1 ! Contents:
;    44          0043  1 !               BUCKET_EMPTY
;    45          0044  1 !               SQUISH_PRIMARY_BUCKET
;    46          0045  1 !               SQUISH_SIDR_BUCKET
;    47          0046  1 !               GET_DOWN_POINTER
;    48          0047  1 !               COMPARE_POINTER
;    49          0048  1 !               SWING_POINTER
;    50          0049  1 !               REMOVE_INDEX_RECORD
;    51          0050  1 !               RECOMPRESS_RECORD
;    52          0051  1 !
;    53          0052  1 !--
;    54          0053  1
;    55          0054  1 !
;    56          0055  1 ! Author:       Peter Lieberwirth       Creation Date:   2-Sep-1981
;    57          0056  1 !
;    58          0057  1 ! Modified by:
;    59          0058  1 !
;    60          0059  1 !       V03-009 TMK0001         Todd M. Katz            03-Feb-1983
;    61          0060  1 !               Add support for Recovery Unit Journalling and RU ROLLBACK
;    62          0061  1 !               Recovery of ISAM files.
;    63          0062  1 !
;    64          0063  1 !               The routine SQUISH_PRIMARY_BUCKET has been modified to squish
;    65          0064  1 !               primary data records that are marked RU_DELETE and re-format
;    66          0065  1 !               primary data records that have been marked RU_UPDATE.
;    67          0066  1 !
;    68          0067  1 !               The routine SQUISH_SIDR_BUCKET has been modified to squish
;    69          0068  1 !               SIDR array elements that are marked RU_DELETE.
;    70          0069  1 !
;    71          0070  1 !               NOTE: The routine SQUISH_SIDR_BUCKET is algorithmically wrong.
;    72          0071  1 !               It doesn't squish out anything! I plan on leaving it the way it
;    73          0072  1 !               is until a massive re-write can be done.
;    74          0073  1 !
;    75          0074  1 !       V03-008 KBT0396         Keith B. Thompson       2-Nov-1982
;    76          0075  1 !               Fix some bugs in squish_primary_bucket and squish_sidr_bucket
;    77          0076  1 !
;    78          0077  1 !       V03-007 KBT0389         Keith B. Thompson       28-Oct-1982
;    79          0078  1 !               Add support for prologue 3 sidrs and do record level
;    80          0079  1 !               space reclamition
;    81          0080  1 !
;    82          0081  1 !       V03-006 KBT0357         Keith B. Thompson       6-Oct-1982
;    83          0082  1 !               Use new merged ctx definitions
;    84          0083  1 !
;    85          0084  1 !       V03-005 KBT0354         Keith B. Thompson       5-Oct-1982
;    86          0085  1 !               Use new linkage definitions
;    87          0086  1 !
```

```
;    88          0087  1 !        V03-004 KBT0049          Keith Thompson          21-Apr-1982
;    89          0088  1 !                Add routine to check for last index record in bucket
;    90          0089  1 !
;    91          0090  1 !        V03-003 KBT0046          Keith Thompson          12-Apr-1982
;    92          0091  1 !                Fix compression bug and increase the key buffers to 257 bytes
;    93          0092  1 !
;    94          0093  1 !        V03-002 KBT0042          Keith Thompson          3-Apr-1982
;    95          0094  1 !                Add routines to compare and swing index pointers
;    96          0095  1 !
;    97          0096  1 !        V03-001 KBT0026          Keith Thompson          29-Mar-1982
;    98          0097  1 !                Do not reclaim data buckets with zero id
;    99          0098  1 !
```

```
;  101          0099  1
;  102          0100  1 LIBRARY 'SYS$LIBRARY:LIB.L32';
;  103          0101  1 LIBRARY 'SRC$:CONVERT';
;  104          0102  1
;  105          0103  1 EXTERNAL ROUTINE
;  106          0104  1         CONV$$RMS_READ_ERROR    : NOVALUE;
;  107          0105  1
;  108          0106  1 FORWARD ROUTINE
;  109          0107  1         SQUISH_PRIMARY_BUCKET   : RL$JSB_REG_9  NOVALUE,
;  110          0108  1         SQUISH_SIDR_BUCKET      : RL$JSB_REG_9  NOVALUE,
;  111          0109  1         RECOMPRESS_RECORD       : RL$JSB_REG_8  NOVALUE;
;  112          0110  1
;  113          0111  1 EXTERNAL
;  114          0112  1         RECL$GL_BUCKET_COUNT,
;  115          0113  1         RECL$GL_SEARCH_BUFFER,
;  116          0114  1         CONV$AB_OUT_FAB         : $FAB_DECL,
;  117          0115  1         CONV$AB_OUT_RAB         : $RAB_DECL;
;  118          0116  1
;  119          0117  1 OWN
;  120          0118  1         INDEX,
;  121          0119  1         VBN_OFFSET,
;  122          0120  1         VBN_FREE_SPACE,
;  123          0121  1         KEY_BUFFER_1    :       BLOCK [ 257, BYTE ],
;  124          0122  1         KEY_BUFFER_2    :       BLOCK [ 257, BYTE ];
;  125          0123  1
```

```
;  127     0124  1 %SBTTL  'BUCKET_EMPTY'
;  128     0125  1 GLOBAL ROUTINE RECL$$BUCKET_EMPTY : RL$JSB_REG_9 =
;  129     0126  1 !++
;  130     0127  1 !
;  131     0128  1 ! Functional Description:
;  132     0129  1 !
;  133     0130  1 !       This routine determines if a bucket is empty.  It handles both
;  134     0131  1 !       index level and data level buckets.
;  135     0132  1 !
;  136     0133  1 ! Calling Sequence:
;  137     0134  1 !
;  138     0135  1 !       RECL$$BUCKET_EMPTY();
;  139     0136  1 !
;  140     0137  1 ! Input Parameters:
;  141     0138  1 !
;  142     0139  1 !       None.
;  143     0140  1 !
;  144     0141  1 ! Implicit Inputs:
;  145     0142  1 !
;  146     0143  1 !       BUCKET          - address of buffer containing bucket
;  147     0144  1 !
;  148     0145  1 ! Output Parameters:
;  149     0146  1 !
;  150     0147  1 !       None.
;  151     0148  1 !
;  152     0149  1 ! Implicit Outputs:
;  153     0150  1 !
;  154     0151  1 !       RECL$GL_BUCKET_COUNT is incremented.
;  155     0152  1 !
;  156     0153  1 ! Routine Value:
;  157     0154  1 !
;  158     0155  1 !       TRUE if bucket is empty
;  159     0156  1 !       FALSE if bucket is not empty or can't be reclaimed
;  160     0157  1 !
;  161     0158  1 ! Routines Called:
;  162     0159  1 !
;  163     0160  1 !       SQUISH_PRIMARY_BUCKET
;  164     0161  1 !       SQUISH_SIDR_BUCKET
;  165     0162  1 !
;  166     0163  1 ! Side Effects:
;  167     0164  1 !
;  168     0165  1 !       None.
;  169     0166  1 !
;  170     0167  1 !--
;  171     0168  1
;  172     0169  2     BEGIN
;  173     0170  2
;  174     0171  2     DEFINE_CTX;
;  175     0172  2     DEFINE_BUCKET;
;  176     0173  2     DEFINE_KEY_DESC;
;  177     0174  2
;  178     0175  2     LITERAL
;  179     0176  2         RECL$_DATA_LEVEL        = 0,
;  180     0177  2         RECL$_BUCKET_EMPTY      = 1,
;  181     0178  2         RECL$_BUCKET_NOT_EMPTY  = 0;
;  182     0179  2
;  183     0180  2     ! Determine if bucket is data level or index level
```

```
;  184          0181  2     !
;  185          0182  2     IF .BUCKET [ BKT$B_LEVEL ] EQLU RECL$_DATA_LEVEL
;  186          0183  2     THEN
;  187          0184  2
;  188          0185  2         ! Determine key
;  189          0186  2         !
;  190          0187  2         IF .KEY_DESC [ KEY$B_KEYREF ] EQL 0
;  191          0188  2         THEN
;  192          0189  2             SQUISH_PRIMARY_BUCKET()
;  193          0190  2         ELSE
;  194          0191  2             SQUISH_SIDR_BUCKET();
;  195          0192  2
;  196          0193  2     ! See if it's empty
;  197          0194  2     !
;  198          0195  2     ! NOTE: Never reclaim the last bucket in a level, due to the complexity of
;  199          0196  2     ! updating high key values in all the levels above.  This is not a serious
;  200          0197  2     ! restriction since most reclamation will be of aging buckets early in
;  201          0198  2     ! collating sequence.
;  202          0199  2     !
;  203          0200  2     IF ( .BUCKET [ BKT$W_KEYFRESPC ] NEQU BKT$C_OVERHDSZ ) OR
;  204          0201  2                                                   .BUCKET [ BKT$V_LASTBKT ]
;  205          0202  2     THEN
;  206          0203  2         RETURN RECL$_BUCKET_NOT_EMPTY
;  207          0204  2     ELSE
;  208          0205  2         RETURN RECL$_BUCKET_EMPTY
;  209          0206  2
;  210          0207  1     END;

                                                        .TITLE  RECL$REC VAX-11 CONVERT/RECLAIM
                                                        .IDENT  \V04-000\

                                                        .PSECT  $OWN$,NOEXE,2

                                                00000 INDEX:    .BLKB   4
                                                00004 VBN_OFFSET:
                                                        .BLKB   4
                                                00008 VBN_FREE_SPACE:
                                                        .BLKB   4
                                                0000C KEY_BUFFER_1:
                                                        .BLKB   257
                                                0010D           .BLKB   3
                                                00110 KEY_BUFFER_2:
                                                        .BLKB   257

                                                        .EXTRN  CONV$$RMS_READ_ERROR
                                                        .EXTRN  RECL$GL_BUCKET_COUNT
                                                        .EXTRN  RECL$GL_SEARCH_BUFFER
                                                        .EXTRN  CONV$AB_OUT_FAB
                                                        .EXTRN  CONV$AB_OUT_RAB

                                                        .PSECT  $CODE$,NOWRT,2

                0C   A9  95 00000 RECL$$BUCKET_EMPTY::
                                                        TSTB    12(BUCKET)              ; 0182
                     0D  12 00003                       BNEQ    2$                      ;
                15   AB  95 00005                       TSTB    21(KEY_DESC)            ; 0187
```

```
                                 05  12 00008          BNEQ    1$                                                      ;
                               0000V 30 0000A          BSBW    SQUISH_PRIMARY_BUCKET                                   ; 0189
                                 03  11 0000D          BRB     2$                                                      ;
                               0000V 30 0000F 1$:      BSBW    SQUISH_SIDR_BUCKET                                      ; 0191
                      0E       04    A9 B1 00012 2$:   CMPW    4(BUCKET), #14                                          ; 0200
                                 04  12 00016          BNEQ    3$                                                      ;
                      03       0D    A9 E9 00018       BLBC    13(BUCKET), 4$                                          ; 0201
                                 50  D4 0001C 3$:      CLRL    R0                                                      ; 0205
                                     05 0001E          RSB                                                             ;
                      50             01  D0 0001F 4$:   MOVL    #1, R0                                                  ;
                                     05 00022          RSB                                                             ; 0207

; Routine Size:  35 bytes,    Routine Base:  $CODE$ + 0000
```

```
;   212     0208  1 %SBTTL  'SQUISH_PRIMARY_BUCKET'
;   213     0209  1 ROUTINE SQUISH_PRIMARY_BUCKET : RL$JSB_REG_9 NOVALUE =
;   214     0210  1 !++
;   215     0211  1 !
;   216     0212  1 ! Functional Description:
;   217     0213  1 !
;   218     0214  1 !       Suqishes the deleted records out of the primary data buckets
;   219     0215  1 !
;   220     0216  1 ! Calling Sequence:
;   221     0217  1 !
;   222     0218  1 !       SQUISH_PRIMARY_BUCKET()
;   223     0219  1 !
;   224     0220  1 ! Input Parameters:
;   225     0221  1 !       None
;   226     0222  1 !
;   227     0223  1 ! Implicit Inputs:
;   228     0224  1 !
;   229     0225  1 !       BUCKET              - address of buffer containing bucket
;   230     0226  1 !       KEY_DESC
;   231     0227  1 !
;   232     0228  1 ! Output Parameters:
;   233     0229  1 !       None
;   234     0230  1 !
;   235     0231  1 ! Implicit Outputs:
;   236     0232  1 !       None
;   237     0233  1 !
;   238     0234  1 ! Routine Value:
;   239     0235  1 !       None
;   240     0236  1 !
;   241     0237  1 ! Routines Called:
;   242     0238  1 !
;   243     0239  1 !       None.
;   244     0240  1 !
;   245     0241  1 ! Side Effects:
;   246     0242  1 !
;   247     0243  1 !       None.
;   248     0244  1 !
;   249     0245  1 !--
;   250     0246  1
;   251     0247  2     BEGIN
;   252     0248  2
;   253     0249  2     DEFINE_BUCKET;
;   254     0250  2     DEFINE_KEY_DESC;
;   255     0251  2
;   256     0252  2     LOCAL
;   257     0253  2         LAST,
;   258     0254  2         POINTER,
;   259     0255  2         RECORD_CTRL     : REF BLOCK [ ,BYTE ];
;   260     0256  2
;   261     0257  2     ! Point to the first record in the bucket
;   262     0258  2     !
;   263     0259  2     POINTER = BKT$K_OVERHDSZ + .BUCKET;
;   264     0260  2
;   265     0261  2     LAST = .POINTER;
;   266     0262  2
;   267     0263  2     ! Count the bucket
;   268     0264  2     !
```

```
;  269          0265  2     RECL$GL_BUCKET_COUNT = .RECL$GL_BUCKET_COUNT + 1;
;  270          0266  2
;  271          0267  2     ! If this bucket has an id of zero then don't bother reclaiming it
;  272          0268  2     !
;  273          0269  2     IF .BUCKET [ BKT$W_NXTRECID ] EQLU 0
;  274          0270  2     THEN
;  275          0271  2         RETURN;
;  276          0272  2
;  277          0273  2     ! Loop untill we have looked at all of the records
;  278          0274  2     !
;  279          0275  3     WHILE .POINTER LSSU ( .BUCKET [ BKT$W_FREESPACE ] + .BUCKET )
;  280          0276  3     DO
;  281          0277  3         BEGIN
;  282          0278  3
;  283          0279  3         ! Point to the control bytes of the record
;  284          0280  3         !
;  285          0281  3         RECORD_CTRL = .POINTER;
;  286          0282  3
;  287          0283  3         ! If this record not deleted check to see if there were any deleted
;  288          0284  3         ! records before it, if so squish them out
;  289          0285  3         !
;  290          0286  4         IF NOT  (.RECORD_CTRL [ IRC$V_DELETED ]
;  291          0287  4                  OR
;  292          0288  4                  .RECORD_CTRL [ IRC$V_RU_DELETE ])
;  293          0289  3         THEN
;  294          0290  4             BEGIN
;  295          0291  4
;  296          0292  4             LOCAL       SQUISH;
;  297          0293  4
;  298          0294  4             ! The current record is not deleted so squish out the
;  299          0295  4             ! deleted ones if there where any
;  300          0296  4             !
;  301          0297  4             SQUISH = .POINTER - .LAST;
;  302          0298  4
;  303          0299  4             IF .SQUISH NEQ 0
;  304          0300  4             THEN
;  305          0301  5                 BEGIN
;  306          0302  5
;  307          0303  5                 LOCAL BYTES;
;  308          0304  5
;  309          0305  5                 ! Number of bytes left in the bucket
;  310          0306  5                 !
;  311          0307  5                 BYTES = ( .BUCKET + .BUCKET [ BKT$W_FREESPACE ] ) - .POINTER;
;  312          0308  5
;  313          0309  5                 ! Move the rest of the records
;  314          0310  5                 !
;  315          0311  5                 CH$MOVE( .BYTES,.POINTER,.LAST );
;  316          0312  5
;  317          0313  5                 ! Update the bucket pointer
;  318          0314  5                 !
;  319          0315  5                 BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
;  320          0316  5                                                                  .SQUISH;
;  321          0317  5
;  322          0318  5                 ! Update our pointers
;  323          0319  5                 !
;  324          0320  5                 POINTER = .POINTER - .SQUISH;
;  325          0321  5                 RECORD_CTRL = .POINTER
```

```
;  326     0322  5
;  327     0323  4                    END;
;  328     0324  4
;  329     0325  4                ! If the current non-deleted primary data record is marked RU_UPDATE
;  330     0326  4                ! then re-format at this time.
;  331     0327  4                !
;  332     0328  4                IF .RECORD_CTRL [ IRC$V_RU_UPDATE ]
;  333     0329  4                THEN
;  334     0330  5                    BEGIN
;  335     0331  5
;  336     0332  5                    LOCAL
;  337     0333  5                        BYTES,
;  338     0334  5                        FAKE_SIZE   : WORD,
;  339     0335  5                        TRUE_SIZE   : WORD;
;  340     0336  5
;  341     0337  5                    ! Turn of the RU_UPDATE bit and retrieve the record's true size
;  342     0338  5                    ! and the number of bytes in the bucket it currently occupies.
;  343     0339  5                    !
;  344     0340  5                    RECORD_CTRL [ IRC$V_RU_UPDATE ] = 0;
;  345     0341  5                    FAKE_SIZE = .RECORD_CTRL [ 9,0,16,0 ];
;  346     0342  5                    TRUE_SIZE = .(.RECORD_CTRL + .FAKE_SIZE + 9)<0,16>;
;  347     0343  5
;  348     0344  5                    ! Place the true size of the primary data record in the size
;  349     0345  5                    ! field of the record overhead, shift the rest of the records
;  350     0346  5                    ! in the bucket to take up the available space, and update the
;  351     0347  5                    ! bucket's freespace offset pointer.
;  352     0348  5                    !
;  353     0349  5                    RECORD_CTRL [ 9,0,16,0 ] = .TRUE_SIZE;
;  354     0350  5
;  355     0351  5                    BYTES = .BUCKET + .BUCKET [ BKT$W_FREESPACE ]
;  356     0352  5                                    - .RECORD_CTRL
;  357     0353  5                                    - .FAKE_SIZE;
;  358     0354  5
;  359     0355  5                    IF .BYTES GTRU 0
;  360     0356  5                    THEN
;  361     0357  5                        CH$MOVE ( .BYTES,
;  362     0358  5                                  .RECORD_CTRL + .FAKE_SIZE,
;  363     0359  5                                  .RECORD_CTRL + .TRUE_SIZE );
;  364     0360  5
;  365     0361  5                    BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ]
;  366     0362  5                                - ( .FAKE_SIZE - .TRUE_SIZE );
;  367     0363  4                    END;
;  368     0364  3                END;
;  369     0365  3
;  370     0366  3            ! Find the next record
;  371     0367  3            !
;  372     0368  3            ! Is this record a RRV record
;  373     0369  3            !
;  374     0370  3            IF .RECORD_CTRL [ IRC$V_RRV ]
;  375     0371  3            THEN
;  376     0372  3
;  377     0373  3                ! If this record has no RRV pointer then set the size to the
;  378     0374  3                ! smallest record there is
;  379     0375  3                !
;  380     0376  3                IF .RECORD_CTRL [ IRC$V_NOPTRSZ ]
;  381     0377  3                THEN
;  382     0378  3
```

```
;  383     0379  1                    ! The least case size of a record is 3 bytes (CTRL and ID)
;  384     0380  1                    !
;  385     0381  1                    POINTER = .POINTER + 3
;  386     0382  1
;  387     0383  1                ELSE
;  388     0384  1
;  389     0385  1                    ! The size of the record with an RRV pointer is
;  390     0386  1                    ! CTRL, ID and Pointer Size (ID and VBN)
;  391     0387  1                    !
;  392     0388  1                    POINTER = .POINTER + 3 + .RECORD_CTRL [ IRC$V_PTRSZ ] + 4
;  393     0389  1
;  394     0390  1            ELSE
;  395     0391  1
;  396     0392  1                ! It is not a RRV, so does it have a size field
;  397     0393  1                !
;  398     0394  1                IF .KEY_DESC [ KEY$V_REC_COMPR ] OR
;  399     0395  1                    .KEY_DESC [ KEY$V_KEY_COMPR ] OR
;  400     0396  1                    ( .CONV$AB_OUT_FAB [ FAB$B_RFM ] EQL FAB$C_VAR )
;  401     0397  1                THEN
;  402     0398  1
;  403     0399  1                    ! Add the size of the record from the size field and control
;  404     0400  1                    !
;  405     0401  1                    POINTER = .POINTER + .RECORD_CTRL [ 9,0,16,0 ] + 11
;  406     0402  1
;  407     0403  1                ELSE
;  408     0404  1
;  409     0405  1                    ! Add the size of the record and control bytes
;  410     0406  1                    !
;  411     0407  1                    POINTER = .POINTER + .CONV$AB_OUT_FAB [ FAB$W_MRS ] + 9;
;  412     0408  1
;  413     0409  1            ! If the last record was not deleted update the last record pointer
;  414     0410  1            !
;  415     0411  1  4         IF NOT  (.RECORD_CTRL [ IRC$V_DELETED ]
;  416     0412  1  4                  OR
;  417     0413  1  4                  .RECORD_CTRL [ IRC$V_RU_DELETE ])
;  418     0414  1            THEN
;  419     0415  1                LAST = .POINTER
;  420     0416  1
;  421     0417  1            END;
;  422     0418  1
;  423     0419  1        ! Update the bucket pointer to catch the last record if it was deleted
;  424     0420  1        !
;  425     0421  1        ! We exit the loop under two cases, 1) the last n records were deleted
;  426     0422  1        ! in which case LAST points to the first deleted record or 2) the last
;  427     0423  1        ! record was not deleted in which case LAST will be pointing to the
;  428     0424  1        ! END of the last record, i.e. same as freespace.
;  429     0425  1        !
;  430     0426  1        BUCKET [ BKT$W_FREESPACE ] = .LAST - .BUCKET;
;  431     0427  1
;  432     0428  1        RETURN
;  433     0429  1
;  434     0430  1    1   END;
```

```
                          05FC   8F  BB 00000 SQUISH_PRIMARY_BUCKET:
                                                  PUSHR   #^M<R2,R3,R4,R5,R6,R7,R8,R10>          ; 0209
                  5E             04  C2 00004     SUBL2   #4, SP                                 ;
                  57        0E   A9  9E 00007     MOVAB   14(R9), POINTER                        ; 0259
                                 57  DD 0000B     PUSHL   POINTER                                ; 0261
                          0000G  CF  D6 0000D     INCL    RECL$GL_BUCKET_COUNT                   ; 0265
                            06   A9  B5 00011     TSTW    6(BUCKET)                              ; 0269
                                 03  12 00014     BNEQ    1$                                     ;
                              00C8   31 00016     BRW     14$                                    ;
             04   AE        04   A9  9E 00019 1$: MOVAB   4(BUCKET), 4(SP)                       ; 0275
                  51        04   BE  3C 0001E 2$: MOVZWL  @4(SP), R1                             ;
       50         51             59  C1 00022     ADDL3   BUCKET, R1, R0                         ;
                  50             57  D1 00026     CMPL    POINTER, R0                            ;
                                 03  1F 00029     BLSSU   3$                                     ;
                              00AE   31 0002B     BRW     13$                                    ;
                  56             57  D0 0002E 3$: MOVL    POINTER, RECORD_CTRL                   ; 0281
       5A         66             02  E0 00031     BBS     #2, (RECORD_CTRL), 6$                  ; 0286
       56         66             05  E0 00035     BBS     #5, (RECORD_CTRL), 6$                  ; 0288
       58         57             6E  C3 00039     SUBL3   LAST, POINTER, SQUISH                  ; 0297
                                 16  13 0003D     BEQL    4$                                     ; 0299
       50         59             51  C1 0003F     ADDL3   R1, BUCKET, R0                         ; 0307
                  50             57  C2 00043     SUBL2   POINTER, BYTES                         ;
  00   BE         67             50  28 00046     MOVC3   BYTES, (POINTER), @LAST                ; 0311
             04   BE             58  A2 0004B     SUBW2   SQUISH, @4(SP)                         ; 0316
                  57             58  C2 0004F     SUBL2   SQUISH, POINTER                        ; 0320
                  56             57  D0 00052     MOVL    POINTER, RECORD_CTRL                   ; 0321
       36         66             06  E1 00055 4$: BBC     #6, (RECORD_CTRL), 6$                  ; 0328
                  66        40   8F  8A 00059     BICB2   #64, (RECORD_CTRL)                     ; 0340
                  51        09   A6  B0 0005D     MOVW    9(RECORD_CTRL), FAKE_SIZE              ; 0341
                  5A             51  3C 00061     MOVZWL  FAKE_SIZE, R10                         ; 0342
       51         56             5A  C1 00064     ADDL3   R10, RECORD_CTRL, R1                   ;
                  50        09   A1  B0 00068     MOVW    9(R1), TRUE_SIZE                       ;
                  58             50  3C 0006C     MOVZWL  TRUE_SIZE, R8                          ; 0349
             09   A6             58  B0 0006F     MOVW    R8, 9(RECORD_CTRL)                     ;
                  50        04   BE  3C 00073     MOVZWL  @4(SP), R0                             ; 0351
                  50             59  C0 00077     ADDL2   BUCKET, R0                             ;
                  50             56  C2 0007A     SUBL2   RECORD_CTRL, R0                        ; 0352
                  50             5A  C2 0007D     SUBL2   R10, BYTES                             ; 0353
                                 05  13 00080     BEQL    5$                                     ; 0355
     6846         61             50  28 00082     MOVC3   BYTES, (R1), (R8)[RECORD_CTRL]         ; 0359
       50         58             5A  C3 00087 5$: SUBL3   R10, R8, R0                            ; 0362
             04   BE             50  A0 0008B     ADDW2   R0, @4(SP)                             ;
       15         66             03  E1 0008F 6$: BBC     #3, (RECORD_CTRL), 8$                  ; 0370
       05         66             04  E1 00093     BBC     #4, (RECORD_CTRL), 7$                  ; 0376
                  57             03  C0 00097     ADDL2   #3, POINTER                            ; 0381
                                 32  11 0009A     BRB     11$                                    ;
50     66         02             00  EF 0009C 7$: EXTZV   #0, #2, (RECORD_CTRL), R0              ; 0388
                  57      07 A047    9E 000A1     MOVAB   7(R0)[POINTER], POINTER                ;
                                 26  11 000A6     BRB     11$                                    ; 0376
                            10   AB  95 000A8 8$: TSTB    16(KEY_DESC)                           ; 0394
                                 0C  19 000AB     BLSS    9$                                     ;
       07    10   AB             06  E0 000AD     BBS     #6, 16(KEY_DESC), 9$                   ; 0395
                  02      0000G  CF  91 000B2     CMPB    CONV$AB_OUT_FAB+31, #2                 ; 0396
                                 0B  12 000B7     BNEQ    10$                                    ;
                  50        09   A6  3C 000B9 9$: MOVZWL  9(RECORD_CTRL), R0                     ; 0401
                  57      0B A047    9E 000BD     MOVAB   11(R0)[POINTER], POINTER               ;
                                 0A  11 000C2     BRB     11$                                    ;
```

```
                                50      0000G   CF  3C 000C4 10$:     MOVZWL  CONV$AB_OUT_FAB+54, R0                  ; 0407
                                57        09 A047  9E 000C9         MOVAB   9(R0)[POINTER], POINTER                 ;
                          07    66          02  E0 000CE 11$:      BBS     #2, (RECORD_CTRL), 12$                  ; 0411
                          03    66          05  E0 000D2           BBS     #5, (RECORD_CTRL), 12$                  ; 0413
                                6E          57  D0 000D6           MOVL    POINTER, LAST                           ; 0415
                                          FF42  31 000D9 12$:      BRW     2$                                      ; 0411
                    04    BE    6E          59  A3 000DC 13$:      SUBW3   BUCKET, LAST, @4(SP)                    ; 0426
                                5E          08  C0 000E1 14$:      ADDL2   #8, SP                                  ; 0430
                                      05FC  8F  BA 000E4           POPR    #^M<R2,R3,R4,R5,R6,R7,R8,R10>           ;
                                            05  000E8             RSB                                             ;

; Routine Size:  233 bytes,    Routine Base:  $CODE$ + 0023
```

```
;  436          0431  1 %SBTTL  'SQUISH_SIDR_BUCKET'
;  437          0432  1 ROUTINE SQUISH_SIDR_BUCKET : RL$JSB_REG_9 NOVALUE =
;  438          0433  1 !++
;  439          0434  1 !
;  440          0435  1 ! Functional Description:
;  441          0436  1 !
;  442          0437  1 !       Suqishes the deleted records out of the sidr data buckets
;  443          0438  1 !
;  444          0439  1 ! Calling Sequence:
;  445          0440  1 !
;  446          0441  1 !       SQUISH_SIDR_BUCKET()
;  447          0442  1 !
;  448          0443  1 ! Input Parameters:
;  449          0444  1 !       None
;  450          0445  1 !
;  451          0446  1 ! Implicit Inputs:
;  452          0447  1 !
;  453          0448  1 !       BUCKET          - address of buffer containing bucket
;  454          0449  1 !       KEY_DESC
;  455          0450  1 !
;  456          0451  1 ! Output Parameters:
;  457          0452  1 !       None
;  458          0453  1 !
;  459          0454  1 ! Implicit Outputs:
;  460          0455  1 !       None
;  461          0456  1 !
;  462          0457  1 ! Routine Value:
;  463          0458  1 !       None
;  464          0459  1 !
;  465          0460  1 ! Routines Called:
;  466          0461  1 !       None
;  467          0462  1 !
;  468          0463  1 ! Side Effects:
;  469          0464  1 !       None
;  470          0465  1 !               NOTE: The routine SQUISH_SIDR_BUCKET is algorithmically wrong.
;  471          0466  1 !               It doesn't squish out anything! I plan on leaving it the way it
;  472          0467  1 !               is until a massive re-write can be done.
;  473          0468  1 !
;  474          0469  1 !--
;  475          0470  1
;  476          0471  2     BEGIN
;  477          0472  2
;  478          0473  2     DEFINE_BUCKET;
;  479          0474  2     DEFINE_KEY_DESC;
;  480          0475  2
;  481          0476  2     LOCAL
;  482          0477  2         LAST,
;  483          0478  2         POINTER         : REF BLOCK [ ,BYTE ],
;  484          0479  2         SIDR            : REF BLOCK [ ,BYTE ];
;  485          0480  2
;  486          0481  2     ! Point to the first record in the bucket
;  487          0482  2     !
;  488          0483  2     SIDR = BKT$K_OVERHDSZ + .BUCKET;
;  489          0484  2
;  490          0485  2     ! Count the bucket
;  491          0486  2     !
;  492          0487  2     RECL$GL_BUCKET_COUNT = .RECL$GL_BUCKET_COUNT + 1;
```

```
;  493     0488  )
;  494     0489      ! Loop untill we have looked at all of the records
;  495     0490      !
;  496     0491      WHILE .SIDR LSSU ( .BUCKET [ BKT$W_FREESPACE ] + .BUCKET )
;  497     0492      DO
;  498     0493          BEGIN
;  499     0494
;  500     0495          ! Point to the first array element
;  501     0496          !
;  502     0497          IF .KEY_DESC [ KEY$V_KEY_COMPR ]
;  503     0498          THEN
;  504     0499              POINTER = .SIDR + .SIDR [ 2,0,8,0 ] + 4
;  505     0500          ELSE
;  506     0501              POINTER = .SIDR + .KEY_DESC [ KEY$B_KEYSZ ] + 2;
;  507     0502
;  508     0503          LAST = .POINTER;
;  509     0504
;  510     0505          ! Loop untill we have looked at all of the array elements
;  511     0506          !
;  512     0507          WHILE .POINTER LSSU ( .SIDR + .SIDR [ 0,0,16,0 ] + 2 )
;  513     0508          DO
;  514     0509
;  515     0510              ! If this array elemented is deleted skip to the next one
;  516     0511              !
;  517     0512              IF  .POINTER [ IRC$V_DELETED ]
;  518     0513                  OR
;  519     0514                  .POINTER [ IRC$V_RU_DELETE ]
;  520     0515              THEN
;  521     0516
;  522     0517                  ! Is there a pointer
;  523     0518                  !
;  524     0519                  IF .POINTER [ IRC$V_NOPTRSZ ]
;  525     0520                  THEN
;  526     0521                      POINTER = .POINTER + 1
;  527     0522                  ELSE
;  528     0523                      POINTER = .POINTER + 1 + .POINTER [ IRC$V_PTRSZ ] + 4
;  529     0524
;  530     0525              ELSE
;  531     0526                  BEGIN
;  532     0527
;  533     0528                  LOCAL   SQUISH;
;  534     0529
;  535     0530                  ! The current sidr is not deleted so squish out the
;  536     0531                  ! deleted ones if there where any
;  537     0532                  !
;  538     0533                  SQUISH = .POINTER - .LAST;
;  539     0534
;  540     0535                  IF .SQUISH NEQ 0
;  541     0536                  THEN
;  542     0537                      BEGIN
;  543     0538
;  544     0539                      LOCAL BYTES;
;  545     0540
;  546     0541                      ! Number of bytes left in the bucket
;  547     0542                      !
;  548     0543                      BYTES = ( .BUCKET + .BUCKET [ BKT$W_FREESPACE ] ) - .POINTER;
;  549     0544
```

```
;   550     0545  5                   ! Move the rest of the records
;   551     0546  5                   !
;   552     0547  5                   CH$MOVE( .BYTES,.POINTER,.LAST );
;   553     0548  5                   !
;   554     0549  5                   ! Update the bucket pointer
;   555     0550  5                   !
;   556     0551  5                   BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
;   557     0552  5                                                .SQUISH;
;   558     0553  5                   !
;   559     0554  5                   ! Update the sidr record size
;   560     0555  5                   !
;   561     0556  5                   SIDR [ 0,0,16,0 ] = .SIDR [ 0,0,16,0 ] - .SQUISH;
;   562     0557  5                   !
;   563     0558  5                   ! Update out pointers
;   564     0559  5                   !
;   565     0560  5                   POINTER = .POINTER - .SQUISH;
;   566     0561  5                   !
;   567     0562  5                   END;
;   568     0563  4               !
;   569     0564  4               ! Find the next sidr element
;   570     0565  4               !
;   571     0566  4               POINTER = .POINTER + 1 + .POINTER [ IRC$V_PTRSZ ] + 4;
;   572     0567  4               !
;   573     0568  4               LAST = .POINTER
;   574     0569  4               !
;   575     0570  4               END;
;   576     0571  3           !
;   577     0572  3           ! Is the sidr array completely deleted
;   578     0573  3           !
;   579     0574  3           IF .POINTER EQL ( .SIDR + .SIDR [ 0,0,16,0 ] )
;   580     0575  4           THEN
;   581     0576  3               BEGIN
;   582     0577  4               !
;   583     0578  4               ! Squish out the entire record (leaving SIDR pointing to the
;   584     0579  4               ! next sidr record)
;   585     0580  4               !
;   586     0581  4               CH$MOVE( .SIDR [ 0,0,16,0 ],.POINTER,.SIDR );
;   587     0582  4               !
;   588     0583  4               ! Update the bucket pointer
;   589     0584  4               !
;   590     0585  4               BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
;   591     0586  4                                                    .SIDR [ 0,0,16,0 ]
;   592     0587  4               !
;   593     0588  4               END
;   594     0589  4           ELSE
;   595     0590  3               !
;   596     0591  3               ! If we don't squish the record find the next one
;   597     0592  3               !
;   598     0593  3               SIDR = .SIDR + .SIDR [ 0,0,16,0 ] + 2
;   599     0594  3               !
;   600     0595  3           END;
;   601     0596  2           !
;   602     0597  2       RETURN
;   603     0598  2       !
;   604     0599  2       END;
;   605     0600  1       !
```

```
                              05FC   8F   BB 00000 SQUISH_SIDR_BUCKET:
                                                            PUSHR   #^M<R2,R3,R4,R5,R6,R7,R8,R10>             ; 0432
                   5E           04   C2 00004               SUBL2   #4, SP                                     ;
                   56      0E   A9   9E 00007               MOVAB   14(R9), SIDR                               ; 0483
                          0000G CF   D6 0000B               INCL    RECL$GL_BUCKET_COUNT                       ; 0487
                   5A      04   A9   9E 0000F               MOVAB   4(BUCKET), R10                             ; 0491
                   50           6A   3C 00013 1$:           MOVZWL  (R10), R0                                  ;
                   50           59   C0 00016               ADDL2   BUCKET, R0                                 ;
                   50           56   D1 00019               CMPL    SIDR, R0                                   ;
                                03   1F 0001C               BLSSU   2$                                         ;
                              0089   31 0001E               BRW     13$                                        ;
      0B       10  AB           06   E1 00021 2$:           BBC     #6, 16(KEY_DESC), 3$                       ; 0497
                   50      02   A6   9A 00026               MOVZBL  2(SIDR), R0                                ; 0499
                   57      04 A046   9E 0002A               MOVAB   4(R0)[SIDR], POINTER                       ;
                                09   11 0002F               BRB     4$                                         ;
                   50      14   AB   9A 00031 3$:           MOVZBL  20(KEY_DESC), R0                           ; 0501
                   57      02 A046   9E 00035               MOVAB   2(R0)[SIDR], POINTER                       ;
                   6E           57   D0 0003A 4$:           MOVL    POINTER, LAST                              ; 0503
                   50           66   3C 0003D 5$:           MOVZWL  (SIDR), R0                                 ; 0507
                   50      02 A046   9E 00040               MOVAB   2(R0)[SIDR], R0                            ;
                   50           57   D1 00045               CMPL    POINTER, R0                                ;
                                45   1E 00048               BGEQU   10$                                        ;
      04           67           02   E0 0004A               BBS     #2, (POINTER), 6$                          ; 0512
      14           67           05   E1 0004E               BBC     #5, (POINTER), 8$                          ; 0514
      04           67           04   E1 00052 6$:           BBC     #4, (POINTER), 7$                          ; 0519
                                57   D6 00056               INCL    POINTER                                    ; 0521
                                E3   11 00058               BRB     5$                                         ;
 50        67      02           00   EF 0005A 7$:           EXTZV   #0, #2, (POINTER), R0                      ; 0523
                   57      05 A047   9E 0005F               MOVAB   5(R0)[POINTER], POINTER                    ;
                                D7   11 00064               BRB     5$                                         ; 0519
      58           57           6E   C3 00066 8$:           SUBL3   LAST, POINTER, SQUISH                      ; 0533
                                17   13 0006A               BEQL    9$                                         ; 0535
                   50           6A   3C 0006C               MOVZWL  (R10), R0                                  ; 0543
                   50           59   C0 0006F               ADDL2   BUCKET, R0                                 ;
                   50           57   C2 00072               SUBL2   POINTER, BYTES                             ;
      00   BE      67           50   28 00075               MOVC3   BYTES, (POINTER), @LAST                    ; 0547
                   6A           58   A2 0007A               SUBW2   SQUISH, (R10)                              ; 0552
                   66           58   A2 0007D               SUBW2   SQUISH, (SIDR)                             ; 0557
                   57           58   C2 00080               SUBL2   SQUISH, POINTER                            ; 0561
 50        67      02           00   EF 00083 9$:           EXTZV   #0, #2, (POINTER), R0                      ; 0567
                   57      05 A047   9E 00088               MOVAB   5(R0)[POINTER], POINTER                    ;
                                AB   11 0008D               BRB     4$                                         ; 0569
                   50           66   3C 0008F 10$:          MOVZWL  (SIDR), R0                                 ; 0575
                   50           56   C0 00092               ADDL2   SIDR, R0                                   ;
                   50           57   D1 00095               CMPL    POINTER, R0                                ;
                                09   12 00098               BNEQ    11$                                        ;
      66           67           66   28 0009A               MOVC3   (SIDR), (POINTER), (SIDR)                  ; 0582
                   6A           66   A2 0009E               SUBW2   (SIDR), (R10)                              ; 0587
                                04   11 000A1               BRB     12$                                        ; 0586
                   56      02   A0   9E 000A3 11$:          MOVAB   2(R0), SIDR                                ; 0594
                              FF69   31 000A7 12$:          BRW     1$                                         ; 0575
                   5E           04   C0 000AA 13$:          ADDL2   #4, SP                                     ; 0600
                              05FC   8F   BA 000AD          POPR    #^M<R2,R3,R4,R5,R6,R7,R8,R10>             ;
```

```
                                                05 000B1         RSB                                                             ;

; Routine Size:  178 bytes,    Routine Base:  $CODE$ + 010C
```

```
;  607     0601  1 %SBTTL  'GET_DOWN_POINTER'
;  608     0602  1 GLOBAL ROUTINE RECL$$GET_DOWN_POINTER ( VBN ) : RL$JSB_REG_8 =
;  609     0603  1 !++
;  610     0604  1 !
;  611     0605  1 ! Functional Description:
;  612     0606  1 !
;  613     0607  1 !       This routine searches the current buffer for the specified
;  614     0608  1 !       down pointer.
;  615     0609  1 !
;  616     0610  1 ! Calling Sequence:
;  617     0611  1 !
;  618     0612  1 !       GET_DOWN_POINTER( VBN );
;  619     0613  1 !
;  620     0614  1 ! Input Parameters:
;  621     0615  1 !
;  622     0616  1 !       VBN     - VBN of bucket on level below being deleted
;  623     0617  1 !
;  624     0618  1 ! Implicit Inputs:
;  625     0619  1 !
;  626     0620  1 !       BUCKET          - address of buffer containing bucket
;  627     0621  1 !       KEY_DESC
;  628     0622  1 !
;  629     0623  1 ! Output Parameters:
;  630     0624  1 !
;  631     0625  1 !       None.
;  632     0626  1 !
;  633     0627  1 ! Implicit Outputs:
;  634     0628  1 !
;  635     0629  1 !   If success:
;  636     0630  1 !
;  637     0631  1 !       INDEX           - number of the index record to remove (0=first)
;  638     0632  1 !       KEY_POINTER     - points to key part to delete
;  639     0633  1 !       KEY_BUFFER_1    - contains the expanded key bucket previous
;  640     0634  1 !                         to one being deleted
;  641     0635  1 !       KEY_BUFFER_2    - contains the expanded key of one being deleted
;  642     0636  1 !
;  643     0637  1 !   If failure the contents of the above registers are undefined.
;  644     0638  1 !
;  645     0639  1 ! Routine Value:
;  646     0640  1 !
;  647     0641  1 !       TRUE if down pointer found, else FALSE
;  648     0642  1 !
;  649     0643  1 ! Routines Called:
;  650     0644  1 !
;  651     0645  1 !       None.
;  652     0646  1 !
;  653     0647  1 ! Side Effects:
;  654     0648  1 !
;  655     0649  1 !       None.
;  656     0650  1 !
;  657     0651  1 !--
;  658     0652  1
;  659     0653  2     BEGIN
;  660     0654  2
;  661     0655  2     DEFINE_CTX;
;  662     0656  2     DEFINE_BUCKET;
;  663     0657  2     DEFINE_KEY_DESC;
```

```
;  664     0658  2     DEFINE_KEY_POINTER;
;  665     0659  2
;  666     0660  2     LOCAL
;  667     0661  2         VBN_OFFSET,
;  668     0662  2         VBN_FREE_SPACE;
;  669     0663  2
;  670     0664  2     ! Initialize the index which counts which record in is the down pointer.
;  671     0665  2     !
;  672     0666  2     INDEX = 0;
;  673     0667  2
;  674     0668  2     ! Initialize offset in bucket to word containing VBN free space pointer
;  675     0669  2     ! so we can get the actual offset to the VBN free space.
;  676     0670  2     !
;  677     0671  2     VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2;
;  678     0672  2
;  679     0673  2     ! Get actual offset of VBN free space.
;  680     0674  2     !
;  681     0675  2     VBN_FREE_SPACE = .BUCKET [ .VBN_OFFSET, 0, 16, 0 ];
;  682     0676  2
;  683     0677  2     ! Now point to first VBN down pointer.
;  684     0678  2     !
;  685     0679  2     VBN_OFFSET = .VBN_OFFSET - ( .BUCKET [ BKT$V_PTR_SZ ] + 2);
;  686     0680  2
;  687     0681  2     ! Scan the VBNs to see if the down pointer is in this bucket.
;  688     0682  2     !
;  689     0683  2     UNTIL .VBN_OFFSET LEQA .VBN_FREE_SPACE
;  690     0684  2     DO
;  691     0685  2
;  692     0686  2         ! Compare the VBN value pointed to by VBN_OFFSET.
;  693     0687  2         !
;  694     0688  2         IF .BUCKET [ .VBN_OFFSET,0,((.BUCKET[ BKT$V_PTR_SZ ] + 2) * 8), 0 ] EQLU
;  695     0689  2                                                                             .VBN
;  696     0690  2         THEN
;  697     0691  2
;  698     0692  2             ! We found the down pointer, so point KEY_POINTER to the key part
;  699     0693  2             ! of the index record.
;  700     0694  2             !
;  701     0695  2             IF .KEY_DESC[ KEY$V_IDX_COMPR ]
;  702     0696  2             THEN
;  703     0697  2                 BEGIN
;  704     0698  2
;  705     0699  2                 ! The key is compressed, so each key part is variable length.
;  706     0700  2                 ! INDEX is currently an index to the right record, so
;  707     0701  2                 ! skip over that many records.
;  708     0702  2                 !
;  709     0703  2                 KEY_POINTER = .BUCKET + BKT$K_OVERHDSZ;
;  710     0704  2
;  711     0705  2                 INCR I FROM 0 TO .INDEX - 1
;  712     0706  2                 DO
;  713     0707  3                     BEGIN
;  714     0708  3
;  715     0709  3                     ! Move the key into the buffer while expanding
;  716     0710  3                     ! the rear end truncation
;  717     0711  3                     !
;  718     0712  3                     !           key_pointer
;  719     0713  3                     !               !
;  720     0714  3                     !               ----------
```

```
            !                 |l|c|      |
            !                 -----------
            !                       \      \ fill \
            !                 --------------------
            ! key_buffer_1    |l|c|  :      :     |
            !                 --------------------
            !                       ^
            !                       |
            !                       |
            !                   filled in when c=0 ( always the first key
            !                                       in the bucket )
            !
            ! CH$COPY( src_len, src, fill, dst_len, dst )

            CH$COPY( .KEY_POINTER [ KEYR$B_LENGTH ],
                     .KEY_POINTER + 2,
                     .( .KEY_POINTER + 1 +
                                .KEY_POINTER [ KEYR$B_LENGTH ] ),
                     .KEY_DESC [ KEY$B_KEYSZ ] -
                                .KEY_POINTER [ KEYR$B_FRONT_COUNT ],
                     KEY_BUFFER_1 + 2 +
                                .KEY_POINTER [ KEYR$B_FRONT_COUNT ] );

            ! Skip to the next key.
            !
            KEY_POINTER = .KEY_POINTER + 2 +
                          .KEY_POINTER [ KEYR$B_LENGTH ]

            END;

        ! Fill in key_buffer_2 with the expanded CURRENT key
        ! first by stuffing the front compresed characters from
        ! the previous key in key_buffer_1 then copy the rest
        ! from the bucket extending it if rear truncation is present
        !
        !             key_pointer
        !                  |
        !                  |
        !             -----------
        !             |l|c|     |
        !             -----------
        !                   \      \
        !             --------      |
        ! key_buffer_1 |l|c|  :     |
        !             --------      |
        !             |      |      | fill \
        !             -------------------
        ! key_buffer_2 |l|c|  :     :      |
        !             -------------------
        !                   ^
        !                   |
        !                   |
        !               filled in when from key_buffer_1
        !               or from .key_pointer when c=0
        !
        ! Fill in the front if there were front compression
        !
        CH$MOVE( .KEY_POINTER [ KEYR$B_FRONT_COUNT ],
                 KEY_BUFFER_1 + 2,
```

```
;  778     0772  3                      KEY_BUFFER_2 + 2);
;  779     0773  3
;  780     0774  3                  ! Copy the rest of the key and expand the rear if neccessary
;  781     0775  3                  !
;  782     0776  3                  CH$COPY( .KEY_POINTER [ KEYR$B_LENGTH ],
;  783     0777  3                           .KEY_POINTER + 2,
;  784     0778  3                           .( .KEY_POINTER + 1 + .KEY_POINTER [ KEYR$B_LENGTH ] ),
;  785     0779  3                           .KEY_DESC [ KEY$B_KEYSZ ] =
;  786     0780  3                                    .KEY_POINTER [ KEYR$B_FRONT_COUNT ],
;  787     0781  3                           KEY_BUFFER_2 + 2 +
;  788     0782  3                                    .KEY_POINTER [ KEYR$B_FRONT_COUNT ] );
;  789     0783  3
;  790     0784  3                  RETURN RECL$_SUCCESS
;  791     0785  3
;  792     0786  3                  END
;  793     0787  3
;  794     0788  3              ELSE
;  795     0789  3
;  796     0790  3                  BEGIN
;  797     0791  3
;  798     0792  3                  ! The key is not compressed, so the key part is fixed length
;  799     0793  3                  ! and easy to find.
;  800     0794  3                  !
;  801     0795  3                  KEY_POINTER = ( .INDEX * .KEY_DESC[ KEY$B_KEYSZ ] )
;  802     0796  3                                + .BUCKET + BKT$K_OVERHDSZ;
;  803     0797  3
;  804     0798  3                  RETURN RECL$_SUCCESS
;  805     0799  3
;  806     0800  3                  END
;  807     0801  3
;  808     0802  3              ELSE
;  809     0803  3                  BEGIN
;  810     0804  3
;  811     0805  3                  ! This was not the down pointer, so get the next down pointer
;  812     0806  3                  !
;  813     0807  3                  VBN_OFFSET = .VBN_OFFSET - ( .BUCKET[ BKT$V_PTR_SZ ] + 2 );
;  814     0808  3                  INDEX = .INDEX + 1;
;  815     0809  3
;  816     0810  3                  END;
;  817     0811  3
;  818     0812  2          ! If we fell through the UNTIL - DO loop it means we didn't find a down
;  819     0813  2          ! pointer.
;  820     0814  2          !
;  821     0815  2          RETURN RECL$_FAILURE
;  822     0816  2
;  823     0817  1          END;


                    00FC   8F  BB 00000 RECL$$GET_DOWN_POINTER::
                                        PUSHR   #^M<R2,R3,R4,R5,R6,R7>                  ; 0602
                         5E             0C  C2 00004         SUBL2   #12, SP             ;
                                 0000'  CF  D4 00007         CLRL    INDEX               ; 0666
                         56        58   AA  3C 0000B         MOVZWL  8(CTX), VBN_OFFSET  ; 0671
                         56             03  C2 0000F         SUBL2   #3, VBN_OFFSET      ;
```

```
                                                 7649  9F 00012          PUSHAB  -(VBN_OFFSET)[BUCKET]                                    ; 0675
                                                    9E  3C 00015          MOVZWL  @(SP)+, VBN_FREE_SPACE
                                                    03  EF 00019          EXTZV   #3, #2, 13(BUCKET), 4(SP)                           ; 0679
                                                    04  AE  C3 00020          SUBL3   4(SP), VBN_OFFSET, R0
                                                    FE  A0  9E 00025          MOVAB   -2(R0), VBN_OFFSET
                                                    56  D1 00029 1$:          CMPL    VBN_OFFSET, VBN_FREE_SPACE                  ; 0683
                                                    03  1A 0002D             BGTRU   2$
                                                  0090  31 0002F             BRW     8$
                                                    03  78 00032 2$:          ASHL    #3, 4(SP), R0                               ; 0688
                                                    10  C0 00037             ADDL2   #16, R0
                                                    00  EF 0003A             EXTZV   #0, R0, (VBN_OFFSET)[BUCKET], R1
                                                    51  D1 00040             CMPL    R1, VBN                                     ; 0689
                                                    6C  12 00044             BNEQ    7$
                                                    03  E1 00046             BBC     #3, 16(KEY_DESC), 5$                        ; 0695
                                                    0E  A9  9E 0004B             MOVAB   14(R9), KEY_POINTER                         ; 0703
                                                    01  CE 0004F             MNEGL   #1, I                                       ; 0705
                                                    21  11 00052             BRB     4$
                                                    68  9A 00054 3$:          MOVZBL  (KEY_POINTER), R2                           ; 0728
                                                    52  C1 00057             ADDL3   R2, KEY_POINTER, R7                         ; 0730
                                                    01  A8  9A 0005B             MOVZBL  1(KEY_POINTER), R0                          ; 0733
                                                    14  AB  9A 0005F             MOVZBL  20(KEY_DESC), R1
                                                    50  C2 00063             SUBL2   R0, R1
                                                    52  2C 00066             MOVC5   R2, 2(KEY_POINTER), 1(R7), R1, -            ; 0734
                                                0000'CF40  0006D                     KEY_BUFFER_1+2[R0]
                                                    02  A7  9E 00071             MOVAB   2(R7), KEY_POINTER                          ; 0739
                                                0000'  CF  F2 00075 4$:          AOBLSS  INDEX, I, 3$
                                                    01  A8  9A 0007B             MOVZBL  1(KEY_POINTER), R7                          ; 0770
                                                    57  28 0007F             MOVC3   R7, KEY_BUFFER_1+2, KEY_BUFFER_2+2          ; 0772
                                                    68  9A 00087             MOVZBL  (KEY_POINTER), R0                           ; 0776
                                                    14  AB  9A 0008A             MOVZBL  20(KEY_DESC), R1                            ; 0780
                                                    57  C2 0008E             SUBL2   R7, R1
                                                    50  2C 00091             MOVC5   R0, 2(KEY_POINTER), 1(R0)[KEY_POINTER], -   ; 0781
                                                0000'CF47  00099                     R1, KEY_BUFFER_2+2[R7]
                                                    0E  11 0009D             BRB     6$                                          ; 0790
                                                    14  AB  9A 0009F 5$:          MOVZBL  20(KEY_DESC), R0                            ; 0795
                                                0000'  CF  C4 000A3             MULL2   INDEX, R0
                                                    0E  A940  9E 000A8             MOVAB   14(BUCKET)[R0], KEY_POINTER                 ; 0796
                                                    01  D0 000AD 6$:          MOVL    #1, R0                                      ; 0798
                                                    12  11 000B0             BRB     9$                                          ; 0790
                                                    04  AE  C3 000B2 7$:          SUBL3   4(SP), VBN_OFFSET, R0                       ; 0807
                                                    FE  A0  9E 000B7             MOVAB   -2(R0), VBN_OFFSET
                                                0000'  CF  D6 000BB             INCL    INDEX                                       ; 0808
                                                  FF67  31 000BF             BRW     1$                                          ; 0688
                                                    50  D4 000C2 8$:          CLRL    R0                                          ; 0815
                                                    0C  C0 000C4 9$:          ADDL2   #12, SP                                     ; 0817
                                                  00FC  8F  BA 000C7             POPR    #^M<R2,R3,R4,R5,R6,R7>
                                                        05 000CB             RSB

; Routine Size:  204 bytes,    Routine Base:  $CODE$ + 01BE
```

```
;  825          0818  1 %SBTTL  'CHECK_LAST'
;  826          0819  1 GLOBAL ROUTINE RECL$$CHECK_LAST : RL$JSB_REG_8 =
;  827          0820  1 !++
;  828          0821  1 !
;  829          0822  1 ! Functional Description:
;  830          0823  1 !
;  831          0824  1 !       This routine checks to see if the current index record
;  832          0825  1 !       indexed by INDEX is the last record in the bucket and if it
;  833          0826  1 !       is the only record
;  834          0827  1 !
;  835          0828  1 ! Calling Sequence:
;  836          0829  1 !
;  837          0830  1 !       CHECK_LAST();
;  838          0831  1 !
;  839          0832  1 ! Input Parameters:
;  840          0833  1 !       none
;  841          0834  1 !
;  842          0835  1 ! Implicit Inputs:
;  843          0836  1 !
;  844          0837  1 !       BUCKET          - address of buffer containing bucket
;  845          0838  1 !       INDEX           - current index record (set by get_down_pointer
;  846          0839  1 !
;  847          0840  1 ! Output Parameters:
;  848          0841  1 !
;  849          0842  1 !       None.
;  850          0843  1 !
;  851          0844  1 ! Implicit Outputs:
;  852          0845  1 !       none
;  853          0846  1 !
;  854          0847  1 ! Routine Value:
;  855          0848  1 !
;  856          0849  1 !       RECL$_SUCCESS   - index record IS the last in bucket and there more
;  857          0850  1 !                         then one record in the bucket
;  858          0851  1 !       RECL$_FAILURE   - index record IS NOT the last in bucket or is the
;  859          0852  1 !                         only one in the bucket
;  860          0853  1 !
;  861          0854  1 ! Routines Called:
;  862          0855  1 !
;  863          0856  1 !       None.
;  864          0857  1 !
;  865          0858  1 ! Side Effects:
;  866          0859  1 !
;  867          0860  1 !       None.
;  868          0861  1 !
;  869          0862  1 !--
;  870          0863  1
;  871          0864  2     BEGIN
;  872          0865  2
;  873          0866  2     DEFINE_CTX;
;  874          0867  2     DEFINE_BUCKET;
;  875          0868  2     DEFINE_KEY_DESC;
;  876          0869  2     DEFINE_KEY_POINTER;
;  877          0870  2
;  878          0871  2     LOCAL
;  879          0872  2         VBN_OFFSET,
;  880          0873  2         LAST_VBN_OFFSET;
;  881          0874  2
```

```
;   882     0875  2     ! We can always reclaim the first record (even if its the last because the
;   883     0876  2     ! whole bucket will then be recalimed)
;   884     0877  2     !
;   885     0878  2     IF .INDEX EQL 0
;   886     0879  2     THEN
;   887     0880  2         RETURN RECL$_FAILURE;
;   888     0881  2
;   889     0882  2     ! Initialize offset in bucket to word containing VBN free space pointer
;   890     0883  2     ! so we can get the actual offset to the VBN free space.
;   891     0884  2     !
;   892     0885  2     VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2;
;   893     0886  2
;   894     0887  2     ! Get actual offset of the last VBN (free_space pointer + 1)
;   895     0888  2     !
;   896     0889  2     LAST_VBN_OFFSET = .BUCKET [ .VBN_OFFSET, 0, 16, 0 ] + 1;
;   897     0890  2
;   898     0891  2     ! Now point to the current VBN down pointer found by get_down_pointer
;   899     0892  2     !
;   900     0893  2     VBN_OFFSET = .VBN_OFFSET -
;   901     0894  2                  ( ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * ( .INDEX + 1 ) );
;   902     0895  2
;   903     0896  2     ! If they are equal then this is the last record in the bucket
;   904     0897  2     !
;   905     0898  2     IF .VBN_OFFSET EQLU .LAST_VBN_OFFSET
;   906     0899  2     THEN
;   907     0900  2         RETURN RECL$_SUCCESS
;   908     0901  2     ELSE
;   909     0902  2         RETURN RECL$_FAILURE
;   910     0903  2
;   911     0904  1     END;


                                0C  BB 00000 RECL$$CHECK_LAST::
                                            PUSHR   #^M<R2,R3>                                  ;  0819
                      50  0000'  CF  D0 00002         MOVL    INDEX, R0                         ;  0878
                                 2A  13 00007         BEQL    1$
                      53    58   AA  3C 00009         MOVZWL  88(CTX), VBN_OFFSET               ;  0885
                      53         03  C2 0000D         SUBL2   #3, VBN_OFFSET
                            7349 9F  9F 00010         PUSHAB  -(VBN_OFFSET)[BUCKET]             ;  0889
                      52         9E  3C 00013         MOVZWL  @(SP)+, LAST_VBN_OFFSET
                                 52  D6 00016         INCL    LAST_VBN_OFFSET
          51    0D    A9         02  EF 00018         EXTZV   #3, #2, T3(BUCKET), R1            ;  0894
                      51         02  C0 0001E         ADDL2   #2, R1
                                 50  D6 00021         INCL    R0
                      50         51  C4 00023         MULL2   R1, R0
                      53         50  C2 00026         SUBL2   R0, VBN_OFFSET
                      52         53  D1 00029         CMPL    VBN_OFFSET, LAST_VBN_OFFSET       ;  0898
                                 05  12 0002C         BNEQ    1$
                      50         01  D0 0002E         MOVL    #1, R0                            ;  0902
                                 02  11 00031         BRB     2$
                                 50  D4 00033 1$:     CLRL    R0                                ;  0904
                                 0C  BA 00035 2$:     POPR    #^M<R2,R3>
                                     05 00037         RSB
```

; Routine Size: 56 bytes,   Routine Base: $CODE$ + 028A

```
;  913          0905  1 %SBTTL  'COMPARE_POINTER'
;  914          0906  1 GLOBAL ROUTINE RECL$$COMPARE_POINTER ( VBN ) : RL$JSB_REG_8 =
;  915          0907  1 !++
;  916          0908  1 !
;  917          0909  1 ! Functional Description:
;  918          0910  1 !
;  919          0911  1 !       This routine compares the next index record pointer in the current
;  920          0912  1 !       buffer for the specified down pointer if necessary is reads in the next
;  921          0913  1 !       bucket in the index chain to get the next index record.
;  922          0914  1 !
;  923          0915  1 ! Calling Sequence:
;  924          0916  1 !
;  925          0917  1 !       COMPARE_POINTER( VBN );
;  926          0918  1 !
;  927          0919  1 ! Input Parameters:
;  928          0920  1 !
;  929          0921  1 !       VBN     - VBN to compare
;  930          0922  1 !
;  931          0923  1 ! Implicit Inputs:
;  932          0924  1 !
;  933          0925  1 !       BUCKET          - address of buffer containing bucket
;  934          0926  1 !       INDEX           - current index record (set by get_down_pointer
;  935          0927  1 !
;  936          0928  1 ! Output Parameters:
;  937          0929  1 !
;  938          0930  1 !       None.
;  939          0931  1 !
;  940          0932  1 ! Implicit Outputs:
;  941          0933  1 !       none
;  942          0934  1 !
;  943          0935  1 ! Routine Value:
;  944          0936  1 !
;  945          0937  1 !       RECL$_SUCCESS   - next index record DOES point to the vbn
;  946          0938  1 !       RECL$_FAILURE   - next index record DOES NOT point to the vbn
;  947          0939  1 !
;  948          0940  1 ! Routines Called:
;  949          0941  1 !
;  950          0942  1 !       None.
;  951          0943  1 !
;  952          0944  1 ! Side Effects:
;  953          0945  1 !
;  954          0946  1 !       None.
;  955          0947  1 !
;  956          0948  1 !--
;  957          0949  1
;  958          0950  2     BEGIN
;  959          0951  2
;  960          0952  2     DEFINE_CTX;
;  961          0953  2     DEFINE_BUCKET;
;  962          0954  2     DEFINE_KEY_DESC;
;  963          0955  2     DEFINE_KEY_POINTER;
;  964          0956  2
;  965          0957  2     LOCAL
;  966          0958  2         VBN_OFFSET,
;  967          0959  2         LAST_VBN_OFFSET,
;  968          0960  2         SEARCH_BUCKET   : REF BLOCK [ ,BYTE ];
;  969          0961  2
```

```
;   970     0962  2
;   971     0963  2        ! Initialize offset in bucket to word containing VBN free space pointer
;   972     0964  2        ! so we can get the actual offset to the VBN free space.
;   973     0965  2        !
;   974     0966  2        VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2;
;   975     0967  2
;   976     0968  2        ! Get actual offset of the last VBN (free_space pointer + 1)
;   977     0969  2        !
;   978     0970  2        LAST_VBN_OFFSET = .BUCKET [ .VBN_OFFSET, 0, 16, 0 ] + 1;
;   979     0971  2
;   980     0972  2        ! Now point to the current VBN down pointer found by get_down_pointer
;   981     0973  2        !
;   982     0974  2        VBN_OFFSET = .VBN_OFFSET -
;   983     0975  2                     ( ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * ( .INDEX + 1 ) );
;   984     0976  2
;   985     0977  2        ! If this is not the end of the pointers then check the next vbn here
;   986     0978  2        ! else read in the next index bucket and search there
;   987     0979  2        !
;   988     0980  2        IF .VBN_OFFSET NEQU .LAST_VBN_OFFSET
;   989     0981  2        THEN
;   990     0982  2            BEGIN
;   991     0983  2
;   992     0984  2            ! Search in the current bucket
;   993     0985  2            !
;   994     0986  2            SEARCH_BUCKET = .BUCKET;
;   995     0987  2
;   996     0988  2            ! Point to the next vbn
;   997     0989  2            !
;   998     0990  2            VBN_OFFSET = .VBN_OFFSET - ( .BUCKET [ BKT$V_PTR_SZ ] + 2 )
;   999     0991  2
;  1000     0992  2            END
;  1001     0993  2        ELSE
;  1002     0994  2
;  1003     0995  2            ! Get the next bucket (if this is the last in the chain return failure)
;  1004     0996  2            !
;  1005     0997  2            IF .BUCKET [ BKT$V_LASTBKT ]
;  1006     0998  2            THEN
;  1007     0999  2                RETURN RECL$_FAILURE
;  1008     1000  2            ELSE
;  1009     1001  2                BEGIN
;  1010     1002  2
;  1011     1003  2                ! Search in the search buffer
;  1012     1004  2                !
;  1013     1005  2                SEARCH_BUCKET = .RECL$GL_SEARCH_BUFFER;
;  1014     1006  2
;  1015     1007  2                ! Read in the next index bucket
;  1016     1008  2                !
;  1017     1009  2                CONV$AB_OUT_RAB [ RAB$L_UBF ] = .SEARCH_BUCKET;
;  1018     1010  2                CONV$AB_OUT_RAB [ RAB$W_USZ ] = .CTX [ CTX$W_BUCKET_SIZE ];
;  1019     1011  2                CONV$AB_OUT_RAB [ RAB$L_BKT ] = .BUCKET [ BKT$L_NXTBKT ];
;  1020     1012  2
;  1021     1013  2                $READ( RAB=CONV$AB_OUT_RAB,ERR=CONV$$RMS_READ_ERROR );
;  1022     1014  2
;  1023     1015  2                ! Point to the first vbn there
;  1024     1016  2                !
;  1025     1017  2                VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2 -
;  1026     1018  2                                    ( .SEARCH_BUCKET [ BKT$V_PTR_SZ ] + 2 )
```

```
4
2            END;
2
2    ! Compare the vbns
2    !
2    IF .VBN EQLU
2       .SEARCH_BUCKET [ .VBN_OFFSET,0,((.SEARCH_BUCKET[ BKT$V_PTR_SZ ]+2)*8),0 ]
2    THEN
2       RETURN RECL$_SUCCESS
2    ELSE
2       RETURN RECL$_FAILURE
2
1    END;



                                                      .EXTRN  SYS$READ

                                1C   BB 00000 RECL$$COMPARE_POINTER::
                                                      PUSHR   #^M<R2,R3,R4>                               ; 0906
                  54       58   AA   3C 00002         MOVZWL  88(CTX), R4                                 ; 0966
                  52            74   DE 00006         MOVAL   -(R4), VBN_OFFSET
                              6249   9F 00009         PUSHAB  (VBN_OFFSET)[BUCKET]                        ; 0970
                  53            9E   3C 0000C         MOVZWL  @(SP)+, LAST_VBN_OFFSET
                                53   D6 0000F         INCL    LAST_VBN_OFFSET
 51      0D   A9  02            03   EF 00011         EXTZV   #3, #2, 13(BUCKET), R1                      ; 0975
                  51            02   C0 00017         ADDL2   #2, R1
          50   0000'  CF        01   C1 0001A         ADDL3   #1, INDEX, R0
                  50            51   C4 00020         MULL2   R1, R0
                  52            50   C2 00023         SUBL2   R0, VBN_OFFSET
                  53            52   D1 00026         CMPL    VBN_OFFSET, LAST_VBN_OFFSET                 ; 0980
                                08   13 00029         BEQL    1$
                  53            59   D0 0002B         MOVL    BUCKET, SEARCH_BUCKET                       ; 0986
                  52            51   C2 0002E         SUBL2   R1, VBN_OFFSET                              ; 0990
                                37   11 00031         BRB     2$
                  50       0D   A9   E8 00033 1$:     BLBS    13(BUCKET), 3$                              ; 0997
                  53    0000G   CF   D0 00037         MOVL    RECL$GL_SEARCH_BUFFER, SEARCH_BUCKET        ; 1005
              0000G  CF         53   D0 0003C         MOVL    SEARCH_BUCKET, CONV$AB_OUT_RAB+36           ; 1009
              0000G  CF    58   AA   B0 00041         MOVW    88(CTX), CONV$AB_OUT_RAB+32                 ; 1010
              0000G  CF    08   A9   D0 00047         MOVL    8(BUCKET), CONV$AB_OUT_RAB+56               ; 1011
                        0000G   CF   9F 0004D         PUSHAB  CONV$$RMS_READ_ERROR                        ; 1013
                        0000G   CF   9F 00051         PUSHAB  CONV$AB_OUT_RAB
          00000000G  00         02   FB 00055         CALLS   #2, SYS$READ
 50      0D   A3  02            03   EF 0005C         EXTZV   #3, #2, 13(SEARCH_BUCKET), R0               ; 1018
              50  54            50   C3 00062         SUBL3   R0, R4, R0
                  52       FE   A0   9E 00066         MOVAB   -2(R0), VBN_OFFSET                          ; 1017
 50      0D   A3  02            03   EF 0006A 2$:     EXTZV   #3, #2, 13(SEARCH_BUCKET), R0               ; 1025
                  50            08   C4 00070         MULL2   #8, R0
                  50            10   C0 00073         ADDL2   #16, R0
 51         6243  50            00   EF 00076         EXTZV   #0, R0, (VBN_OFFSET)[SEARCH_BUCKET], R1
                  51       10   AE   D1 0007C         CMPL    VBN, R1
                                05   12 00080         BNEQ    3$
                  50            01   D0 00082         MOVL    #1, R0                                      ; 1029
                                02   11 00085         BRB     4$
                                50   D4 00087 3$:     CLRL    R0                                          ; 1031
                                1C   BA 00089 4$:     POPR    #^M<R2,R3,R4>
                                     05 0008B         RSB
```

```
; Routine Size:  140 bytes,    Routine Base:  $CODE$ + 02C2
```

```
;  1041     1032  1 %SBTTL  'SWING_POINTER'
;  1042     1033  1 GLOBAL ROUTINE RECL$$SWING_POINTER ( VBN ) : RL$JSB_REG_8 NOVALUE =
;  1043     1034  1 !++
;  1044     1035  1 !
;  1045     1036  1 ! Functional Description:
;  1046     1037  1 !
;  1047     1038  1 !       This routine will stuff the VBN into the curretn index record
;  1048     1039  1 !
;  1049     1040  1 ! Calling Sequence:
;  1050     1041  1 !
;  1051     1042  1 !       SWING_POINTER( VBN );
;  1052     1043  1 !
;  1053     1044  1 ! Input Parameters:
;  1054     1045  1 !
;  1055     1046  1 !       VBN     - VBN to stuff
;  1056     1047  1 !
;  1057     1048  1 ! Implicit Inputs:
;  1058     1049  1 !
;  1059     1050  1 !       BUCKET          - address of buffer containing bucket
;  1060     1051  1 !       INDEX           - index record to stuff
;  1061     1052  1 !
;  1062     1053  1 ! Output Parameters:
;  1063     1054  1 !
;  1064     1055  1 !       None.
;  1065     1056  1 !
;  1066     1057  1 ! Implicit Outputs:
;  1067     1058  1 !       none
;  1068     1059  1 !
;  1069     1060  1 ! Routine Value:
;  1070     1061  1 !       none
;  1071     1062  1 !
;  1072     1063  1 ! Routines Called:
;  1073     1064  1 !
;  1074     1065  1 !       None.
;  1075     1066  1 !
;  1076     1067  1 ! Side Effects:
;  1077     1068  1 !
;  1078     1069  1 !       None.
;  1079     1070  1 !
;  1080     1071  1 !--
;  1081     1072  1
;  1082     1073  2     BEGIN
;  1083     1074  2
;  1084     1075  2     DEFINE_CTX;
;  1085     1076  2     DEFINE_BUCKET;
;  1086     1077  2     DEFINE_KEY_DESC;
;  1087     1078  2     DEFINE_KEY_POINTER;
;  1088     1079  2
;  1089     1080  2     LOCAL
;  1090     1081  2         VBN_OFFSET;
;  1091     1082  2
;  1092     1083  2     ! Point to current VBN down pointer
;  1093     1084  2     ! Which is: Bucket size - 2 bytes for check and spare - 2 bytes for
;  1094     1085  2     !               vbn freespace pointer - index into the array
;  1095     1086  2     !
;  1096     1087  2     VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2 -
;  1097     1088  2                  ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * ( .INDEX + 1 );
```

```
; 1098            1089  2
; 1099            1090  2      ! Stuff the vbn
; 1100            1091  2      !
; 1101            1092  2      BUCKET [ .VBN_OFFSET,0,( ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * 8 ),0 ] = .VBN;
; 1102            1093  2
; 1103            1094  2      RETURN
; 1104            1095  2
; 1105            1096  1      END;



                                                      0C   BB 00000 RECL$$SWING_POINTER::
                                                                              PUSHR    #^M<R2,R3>                              ; 1033
       52       0D    A9            02              03   EF 00002           EXTZV    #3, #2, 13(BUCKET), R2                  ; 1088
                                    51       02     A2   9E 00008           MOVAB    2(R2), R1                               ;
                      50    0000'   CF              01   C1 0000C           ADDL3    #1, INDEX, R0                           ;
                                    50              51   C4 00012           MULL2    R1, R0                                  ;
                                    53       58     AA   3C 00015           MOVZWL   88(CTX), R3                             ;
                      50            53              50   C3 00019           SUBL3    R0, R3, R0                              ;
                                    50              03   C2 0001D           SUBL2    #3, VBN_OFFSET                          ; 1087
                      51            52              03   78 00020           ASHL     #3, R2, R1                              ; 1092
                                    51              10   C0 00024           ADDL2    #16, R1                                 ;
     7049             51            00       0C     AE   F0 00027           INSV     VBN, #0, R1, -(VBN_OFFSET)[BUCKET]      ;
                                                    0C   BA 0002E           POPR     #^M<R2,R3>                              ; 1096
                                                         05 00030           RSB                                              ;

; Routine Size:  49 bytes,    Routine Base:  $CODE$ + 034E
```

```
;  1107          1097  1 %SBTTL  'REMOVE_INDEX_RECORD'
;  1108          1098  1 GLOBAL ROUTINE RECL$$REMOVE_INDEX_RECORD : RL$JSB_REG_8 NOVALUE =
;  1109          1099  1 !++
;  1110          1100  1 !
;  1111          1101  1 ! Functional Description:
;  1112          1102  1 !
;  1113          1103  1 !       This routine actually squishes out the index record from the index
;  1114          1104  1 !       bucket.
;  1115          1105  1 !
;  1116          1106  1 ! Calling Sequence:
;  1117          1107  1 !
;  1118          1108  1 !       REMOVE_INDEX_RECORD();
;  1119          1109  1 !
;  1120          1110  1 ! Input Parameters:
;  1121          1111  1 !
;  1122          1112  1 !       None.
;  1123          1113  1 !
;  1124          1114  1 ! Implicit Inputs:
;  1125          1115  1 !
;  1126          1116  1 !       INDEX           - number of the index record to remove
;  1127          1117  1 !       KEY_POINTER     - points to key part of index record to remove
;  1128          1118  1 !       KEY_BUFFER_1    - contains fully expanded previous key
;  1129          1119  1 !       KEY_BUFFER_2    - contains fully expanded current key
;  1130          1120  1 !       BUCKET          - points to buffer containing bucket
;  1131          1121  1 !
;  1132          1122  1 ! Output Parameters:
;  1133          1123  1 !
;  1134          1124  1 !       None.
;  1135          1125  1 !
;  1136          1126  1 ! Implicit Outputs:
;  1137          1127  1 !
;  1138          1128  1 !       Index bucket has more freespace, since a record was squished out.
;  1139          1129  1 !
;  1140          1130  1 ! Routine Value:
;  1141          1131  1 !
;  1142          1132  1 !       None.
;  1143          1133  1 !
;  1144          1134  1 ! Routines Called:
;  1145          1135  1 !
;  1146          1136  1 !       RECOMPRESS_RECORD
;  1147          1137  1 !
;  1148          1138  1 ! Side Effects:
;  1149          1139  1 !
;  1150          1140  1 !       None.
;  1151          1141  1 !
;  1152          1142  1 !--
;  1153          1143  1
;  1154          1144  2     BEGIN
;  1155          1145  2
;  1156          1146  2     DEFINE_CTX;
;  1157          1147  2     DEFINE_BUCKET;
;  1158          1148  2     DEFINE_KEY_DESC;
;  1159          1149  2     DEFINE_KEY_POINTER;
;  1160          1150  2
;  1161          1151  2     !++
;  1162          1152  2     !
;  1163          1153  2     ! Squish out the VBN part of the index record
```

```
;  1164          1154  2     !
;  1165          1155  2     !--
;  1166          1156  2     !
;  1167          1157  2     BEGIN
;  1168          1158  2     !
;  1169          1159  2     LOCAL
;  1170          1160  2         OFFSET,             ! Offset to the vbn freespace pointer
;  1171          1161  2         VBN_SIZE,           ! Size of vbn in bytes
;  1172          1162  2         BITS,               ! Size of vbn in bits
;  1173          1163  2         FREESPACE,          ! Pointer offset to the top of the vbns
;  1174          1164  2         VBN,                ! Pointer offset to the vbn to remove
;  1175          1165  2         SOURCE,             ! Pointer offset to the Source
;  1176          1166  2         DEST;               ! Pointer offset to the Destination
;  1177          1167  2     !
;  1178          1168  2     ! Find the offset to the vbn freespace pointer
;  1179          1169  2     !
;  1180          1170  2     OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 4;
;  1181          1171  2     !
;  1182          1172  2     ! Get the size of the vbns in bytes
;  1183          1173  2     !
;  1184          1174  2     VBN_SIZE = .BUCKET [ BKT$V_PTR_SZ ] + 2;
;  1185          1175  2     !
;  1186          1176  2     ! Now get it in bits
;  1187          1177  2     !
;  1188          1178  2     BITS = .VBN_SIZE * 8;
;  1189          1179  2     !
;  1190          1180  2     ! Find the top the vbns
;  1191          1181  2     !
;  1192          1182  2     FREESPACE = .BUCKET [ .OFFSET,0,16,0 ];
;  1193          1183  2     !
;  1194          1184  2     ! Find the vbn we want to remove
;  1195          1185  2     !
;  1196          1186  2     VBN = .OFFSET - ( .VBN_SIZE * ( .INDEX + 1 ) );
;  1197          1187  2     !
;  1198          1188  2     ! Set up the destindtion
;  1199          1189  2     !
;  1200          1190  2     DEST = .VBN;
;  1201          1191  2     !
;  1202          1192  2     ! And the source
;  1203          1193  2     !
;  1204          1194  2     SOURCE = .DEST - .VBN_SIZE;
;  1205          1195  2     !
;  1206          1196  2     ! Do each vbn
;  1207          1197  2     !
;  1208          1198  2     WHILE .SOURCE GEQU .FREESPACE
;  1209          1199  3     DO
;  1210          1200  4         BEGIN
;  1211          1201  4         !
;  1212          1202  4         ! Copy the vbn to the new location
;  1213          1203  4         !
;  1214          1204  4         BUCKET [ .DEST,0,.BITS,0 ] = .BUCKET [ .SOURCE,0,.BITS,0 ];
;  1215          1205  4         !
;  1216          1206  4         ! Update the pointers
;  1217          1207  4         !
;  1218          1208  4         DEST = .DEST - .VBN_SIZE;
;  1219          1209  4         SOURCE = .SOURCE - .VBN_SIZE
;  1220          1210  4
```

```
;  1221     1211  3          END;
;  1222     1212
;  1223     1213  3      ! Update the freespace pointer in the bucket
;  1224     1214  3      !
;  1225     1215  3      BUCKET [ .OFFSET,0,16,0 ] = .FREESPACE + .VBN_SIZE;
;  1226     1216  3
;  1227     1217  3      ! If freespace pointer points to the bottom of the bucket it is
;  1228     1218  3      ! empty so don't bother to fool with the data part (but do set
;  1229     1219  3      ! the keyfreespace pointer)
;  1230     1220  3      !
;  1231     1221  3      IF .BUCKET [ .OFFSET,0,16,0 ] EQLU .OFFSET
;  1232     1222  3      THEN
;  1233     1223  3          BEGIN
;  1234     1224  4          BUCKET[ BKT$W_KEYFRESPC ] = BKT$C_OVERHDSZ;
;  1235     1225  4          RETURN
;  1236     1226  4          END
;  1237     1227  3
;  1238     1228  3      END;
;  1239     1229  3
;  1240     1230  3      !++
;  1241     1231  3      !
;  1242     1232  3      ! Squeeze out the KEY part of the index record
;  1243     1233  3      !
;  1244     1234  3      !--
;  1245     1235  3
;  1246     1236  3      BEGIN
;  1247     1237  3
;  1248     1238  3      LOCAL
;  1249     1239  3          DELETE_SIZE;
;  1250     1240  3
;  1251     1241  3      ! Calculate from address and size for squish differently if index is
;  1252     1242  3      ! compressed or not.  Also do KEYFRESPC depending on index compression.
;  1253     1243  3      !
;  1254     1244  3      IF .KEY_DESC[ KEY$V_IDX_COMPR ]
;  1255     1245  3      THEN
;  1256     1246  4          BEGIN
;  1257     1247  4
;  1258     1248  4          LOCAL
;  1259     1249  4              NEXT        : REF BLOCK [ ,BYTE ];  ! Pointer to the next key
;  1260     1250  4                                                  ! to replace the deleted one
;  1261     1251  4
;  1262     1252  4          ! The size of the deleted space is size of the old record MINUS
;  1263     1253  4          ! the DIFFERENCE between the size of next record before compression
;  1264     1254  4          ! and the size of it after compression.
;  1265     1255  4          ! First save the size of old record.
;  1266     1256  4          !
;  1267     1257  4          DELETE_SIZE = .KEY_POINTER [ KEYR$B_LENGTH ] + 2;
;  1268     1258  4
;  1269     1259  4          ! Next thing to do is recompress the next record after the current
;  1270     1260  4          ! one we start by coping it into key_buffer_2 (where the to-be-deleted
;  1271     1261  4          ! key is)
;  1272     1262  4          !
;  1273     1263  4          NEXT = .KEY_POINTER + .KEY_POINTER [ KEYR$B_LENGTH ] + 2;
;  1274     1264  4
;  1275     1265  4          ! If there IS a next key then copy it and compress it
;  1276     1266  4          !
;  1277     1267  5          IF .NEXT LSSU ( .BUCKET + .BUCKET [ BKT$W_KEYFRESPC ] )
```

```
;  1278     1268  4          THEN
;  1279     1269  4              BEGIN
;  1280     1270  4
;  1281     1271  4              LOCAL OLD_SIZE;
;  1282     1272  4
;  1283     1273  4              ! Save the old size of the next record
;  1284     1274  4              !
;  1285     1275  4              OLD_SIZE = .NEXT [ KEYR$B_LENGTH ];
;  1286     1276  4
;  1287     1277  4              ! Copy the next key while expanding the rear
;  1288     1278  4              !
;  1289     1279  4              CH$COPY( .NEXT [ KEYR$B_LENGTH ],
;  1290     1280  4                       .NEXT + 2,
;  1291     1281  4                       .( .NEXT + 1 + .NEXT [ KEYR$B_LENGTH ] ),
;  1292     1282  4                       .KEY_DESC [ KEY$B_KEYSZ ] - .NEXT [ KEYR$B_FRONT_COUNT ],
;  1293     1283  4                       KEY_BUFFER_2 + 2 + .NEXT [ KEYR$B_FRONT_COUNT ] );
;  1294     1284  4
;  1295     1285  4              ! Recompress the new key in key_buffer_2
;  1296     1286  4              !
;  1297     1287  4              RECOMPRESS_RECORD();
;  1298     1288  4
;  1299     1289  4              ! Key_buffer_2 now contains a compressed key (w/control info)
;  1300     1290  4              ! so move it into the bucket
;  1301     1291  4              !
;  1302     1292  4              CH$MOVE( .KEY_BUFFER_2 [ KEYR$B_LENGTH ] + 2,
;  1303     1293  4                       KEY_BUFFER_2,
;  1304     1294  4                       .KEY_POINTER );
;  1305     1295  4
;  1306     1296  4              ! Now we can figure the ammount of space deleted
;  1307     1297  4              !
;  1308     1298  4              DELETE_SIZE = .DELETE_SIZE -
;  1309     1299  4                            ( .KEY_POINTER [ KEYR$B_LENGTH ] - .OLD_SIZE );
;  1310     1300  4
;  1311     1301  4              ! We must now move the rest of the keys in the bucket
;  1312     1302  4              !
;  1313     1303  4              CH$MOVE( ( .BUCKET + .BUCKET [ BKT$W_KEYFRESPC ] ) -
;  1314     1304  4                                   ( .NEXT + .NEXT [ KEYR$B_LENGTH ] + 2 ),
;  1315     1305  4                       .NEXT + .NEXT [ KEYR$B_LENGTH ] + 2,
;  1316     1306  4                       .KEY_POINTER + .KEY_POINTER [ KEYR$B_LENGTH ] + 2 )
;  1317     1307  4
;  1318     1308  4              END;
;  1319     1309  4          END
;  1320     1310  4
;  1321     1311  3      ELSE
;  1322     1312  4          BEGIN
;  1323     1313  4
;  1324     1314  4          ! Set the delete size
;  1325     1315  4          !
;  1326     1316  4          DELETE_SIZE = .KEY_DESC [ KEY$B_KEYSZ ];
;  1327     1317  4
;  1328     1318  4          ! Move the rest of the keys
;  1329     1319  4          !
;  1330     1320  4          CH$MOVE( ( .BUCKET + .BUCKET [ BKT$W_KEYFRESPC ] ) -
;  1331     1321  4                                ( .KEY_POINTER + .KEY_DESC [ KEY$B_KEYSZ ] ),
;  1332     1322  4                   .KEY_POINTER + .KEY_DESC [ KEY$B_KEYSZ ],
;  1333     1323  4                   .KEY_POINTER )
;  1334     1324  4
```

```
1325 3          END;
1326 3
1327 3      ! Update KEYFRESPC since we squished out a key
1328 3      !
1329 3      BUCKET [ BKT$W_KEYFRESPC ] = .BUCKET [ BKT$W_KEYFRESPC ] - .DELETE_SIZE
1330 3
1331 2      END;
1332 2
1333 2      RETURN
1334 2
1335 1      END;



                                  00FC   8F  BB 00000 RECL$$REMOVE_INDEX_RECORD::
                                                             PUSHR   #^M<R2,R3,R4,R5,R6,R7>                    ; 1098
                            5E           04  C2 00004        SUBL2   #4, SP
                            51       58  AA  3C 00007        MOVZWL  88(CTX), OFFSET                           ; 1170
                            51           03  C2 0000B        SUBL2   #3, OFFSET
   52       0D   A9         02           03  EF 0000E        EXTZV   #3, #2, 13(BUCKET), VBN_SIZE              ; 1174
                            52           02  C0 00014        ADDL2   #2, VBN_SIZE
                 56         52           03  78 00017        ASHL    #3, VBN_SIZE, BITS                        ; 1178
                                       7149  9F 0001B        PUSHAB  -(OFFSET)[BUCKET]                         ; 1182
                            55           9E  3C 0001E        MOVZWL  @(SP)+, FREESPACE
                 50  0000'  CF           01  C1 00021        ADDL3   #1, INDEX, R0                             ; 1186
                            50           52  C4 00027        MULL2   VBN_SIZE, R0
                 50         51           50  C3 0002A        SUBL3   R0, OFFSET, VBN
                            53           50  D0 0002E        MOVL    VBN, DEST                                 ; 1190
                 54         53           52  C3 00031        SUBL3   VBN_SIZE, DEST, SOURCE                    ; 1194
                            55           54  D1 00035 1$:    CMPL    SOURCE, FREESPACE                         ; 1198
                                         14  1F 00038        BLSSU   2$
   50          6449         56           00  EF 0003A        EXTZV   #0, BITS, (SOURCE)[BUCKET], R0            ; 1204
 6349            56         00           50  F0 00040        INSV    R0, #0, BITS, (DEST)[BUCKET]
                            53           52  C2 00046        SUBL2   VBN_SIZE, DEST                            ; 1208
                            54           52  C2 00049        SUBL2   VBN_SIZE, SOURCE                          ; 1209
                                         E7  11 0004C        BRB     1$
                                       6149  9F 0004E 2$:    PUSHAB  (OFFSET)[BUCKET]                          ; 1215
                 9E         55           52  A1 00051        ADDW3   VBN_SIZE, FREESPACE, @(SP)+
   51          6149         10           00  ED 00055        CMPZV   #0, #16, (OFFSET)[BUCKET], OFFSET         ; 1221
                                         07  12 0005B        BNEQ    3$
                       04   A9           0E  B0 0005D        MOVW    #14, 4(BUCKET)                            ; 1224
                                       0091  31 00061        BRW     6$                                        ; 1223
                 6E    10   AB           03  E1 00064 3$:    BBC     #3, 16(KEY_DESC), 4$                      ; 1244
                            50           68  9A 00069        MOVZBL  (KEY_POINTER), R0                         ; 1257
                            57       02  A0  9E 0006C        MOVAB   2(R0), DELETE_SIZE
                            51     02 A048   9E 00070        MOVAB   2(R0)[KEY_POINTER], NEXT                  ; 1263
                            50       04  A9  3C 00075        MOVZWL  4(BUCKET), R0                             ; 1267
                            50           59  C0 00079        ADDL2   BUCKET, R0
                            50           51  D1 0007C        CMPL    NEXT, R0
                                         70  1E 0007F        BGEQU   5$
                            53           61  9A 00081        MOVZBL  (NEXT), R3                                ; 1275
                            56           53  D0 00084        MOVL    R3, OLD_SIZE
                 6E         51           53  C1 00087        ADDL3   R3, NEXT, (SP)                            ; 1281
                            50       01  A1  9A 0008B        MOVZBL  1(NEXT), R0                               ; 1282
                            52       14  AB  9A 0008F        MOVZBL  20(KEY_DESC), R2
```

```
                                   52         50  C2 00093         SUBL2   R0, R2                                      ;
                        7E         6E         01  C1 00096         ADDL3   #1, (SP), -(SP)                             ;  1283
        52              9E    02   A1         53  2C 0009A         MOVC5   R3, 2(NEXT), @(SP)+, R2, KEY_BUFFER_2+2[R0] ;
                                        0000'CF40    000A0                                                             ;
                                          0000V  30  000A4         BSBW    RECOMPRESS_RECORD                           ;  1287
                                   50   0000'  CF  9A 000A7        MOVZBL  KEY_BUFFER_2, R0                            ;  1292
                                   50          02  C0 000AC        ADDL2   #2, R0                                      ;
                        68   0000' CF          50  28 000AF        MOVC3   R0, KEY_BUFFER_2, (KEY_POINTER)             ;  1294
                                   51          68  9A 000B5        MOVZBL  (KEY_POINTER), R1                           ;  1299
                                   56          51  C2 000B8        SUBL2   R1, R6                                      ;
                                   57          56  C0 000BB        ADDL2   R6, DELETE_SIZE                             ;
                                   50     04   A9  3C 000BE        MOVZWL  4(BUCKET), R0                               ;  1303
                                   50          59  C0 000C2        ADDL2   BUCKET, R0                                  ;
                                   50          6E  C2 000C5        SUBL2   (SP), R0                                    ;  1304
                                   50          02  C2 000C8        SUBL2   #2, R0                                      ;  1303
                                   52          6E  D0 000CB        MOVL    (SP), R2                                    ;  1305
                   02 A148    02   A2          50  28 000CE        MOVC3   R0, 2(R2), 2(R1)[KEY_POINTER]               ;  1306
                                               1A  11 000D5        BRB     5$                                          ;  1244
                                   52     14   AB  9A 000D7  4$:   MOVZBL  20(KEY_DESC), R2                            ;  1316
                                   57          52  D0 000DB        MOVL    R2, DELETE_SIZE                             ;
                                   50     04   A9  3C 000DE        MOVZWL  4(BUCKET), R0                               ;  1320
                        51         59          50  C1 000E2        ADDL3   R0, BUCKET, R1                              ;
                        50         58          52  C1 000E6        ADDL3   R2, KEY_POINTER, R0                         ;  1321
                                   51          50  C2 000EA        SUBL2   R0, R1                                      ;
                        68         60          51  28 000ED        MOVC3   R1, (R0), (KEY_POINTER)                     ;  1323
                              04   A9          57  A2 000F1  5$:   SUBW2   DELETE_SIZE, 4(BUCKET)                      ;  1329
                                   5E          04  C0 000F5  6$:   ADDL2   #4, SP                                      ;  1335
                                     00FC      8F  BA 000F8        POPR    #^M<R2,R3,R4,R5,R6,R7>                      ;
                                                   05 000FC        RSB                                                 ;

; Routine Size:  253 bytes,    Routine Base:  $CODE$ + 037F


; 1346          1336  1
```

```
;  1348     1337  1 %SBTTL  'RECOMPRESS_RECORD'
;  1349     1338  1 ROUTINE RECOMPRESS_RECORD : RL$JSB_REG_8 NOVALUE =
;  1350     1339  1 !++
;  1351     1340  1 !
;  1352     1341  1 ! Functional Description:
;  1353     1342  1 !
;  1354     1343  1 !       This routine will recompress the index record in key_buffer_2
;  1355     1344  1 !
;  1356     1345  1 ! Calling Sequence:
;  1357     1346  1 !
;  1358     1347  1 !       RECOMPRESS_RECORD()
;  1359     1348  1 !
;  1360     1349  1 ! Input Parameters:
;  1361     1350  1 !
;  1362     1351  1 !       None.
;  1363     1352  1 !
;  1364     1353  1 ! Implicit Inputs:
;  1365     1354  1 !
;  1366     1355  1 !       KEY_BUFFER_1    - contains expanded key to base re-compression upon
;  1367     1356  1 !       KEY_BUFFER_2    - contains expanded key to re-compress
;  1368     1357  1 !
;  1369     1358  1 ! Output Parameters:
;  1370     1359  1 !
;  1371     1360  1 !       None.
;  1372     1361  1 !
;  1373     1362  1 ! Implicit Outputs:
;  1374     1363  1 !
;  1375     1364  1 !       None.
;  1376     1365  1 !
;  1377     1366  1 ! Routine Value:
;  1378     1367  1 !
;  1379     1368  1 !       None.
;  1380     1369  1 !
;  1381     1370  1 ! Routines Called:
;  1382     1371  1 !
;  1383     1372  1 !       None.
;  1384     1373  1 !
;  1385     1374  1 ! Side Effects:
;  1386     1375  1 !
;  1387     1376  1 !       Index record in key_buffer_2 is compressed.
;  1388     1377  1 !
;  1389     1378  1 !--
;  1390     1379  1
;  1391     1380  2     BEGIN
;  1392     1381  2
;  1393     1382  2     DEFINE_CTX;
;  1394     1383  2     DEFINE_BUCKET;
;  1395     1384  2     DEFINE_KEY_DESC;
;  1396     1385  2     DEFINE_KEY_POINTER;
;  1397     1386  2
;  1398     1387  2     BIND
;  1399     1388  2         KEY_1 = KEY_BUFFER_1 + 2 : VECTOR [ ,BYTE ],     ! Key part of the record
;  1400     1389  2         KEY_2 = KEY_BUFFER_2 + 2 : VECTOR [ ,BYTE ];
;  1401     1390  2
;  1402     1391  2     LOCAL
;  1403     1392  2         LENGTH;
;  1404     1393  2
```

```
;  1405          1394  2      ! Assume no compression
;  1406          1395  2      !
;  1407          1396  2      KEY_BUFFER_2 [ KEYR$B_FRONT_COUNT ] = 0;
;  1408          1397  2
;  1409          1398  2      LENGTH = .KEY_DESC [ KEY$B_KEYSZ ];
;  1410          1399  2
;  1411          1400  2      ! If this is NOT the first key in the bucket do front compression
;  1412          1401  2      !
;  1413          1402  2      IF .INDEX NEQU 0
;  1414          1403  2      THEN
;  1415          1404  2
;  1416          1405  2          ! Find the first position where the two keys differ
;  1417          1406  2          !
;  1418          1407  2          INCR I FROM 0 TO ( .LENGTH - 1 ) BY 1
;  1419          1408  2          DO
;  1420          1409  2
;  1421          1410  2              ! If the characters are not equal we found the end
;  1422          1411  2              !
;  1423          1412  2              IF ( .KEY_1 [ .I ] NEQU .KEY_2 [ .I ] )
;  1424          1413  2              THEN
;  1425          1414  2                  BEGIN
;  1426          1415  2
;  1427          1416  2                  ! I is now the number of compressed characters
;  1428          1417  2                  !
;  1429          1418  2                  KEY_BUFFER_2 [ KEYR$B_FRONT_COUNT ] = .I;
;  1430          1419  2
;  1431          1420  2                  ! Shorten the length
;  1432          1421  2                  !
;  1433          1422  2                  LENGTH = .LENGTH - .I;
;  1434          1423  2
;  1435          1424  2                  ! If there was some compression move the key a little
;  1436          1425  2                  !
;  1437          1426  2                  IF .I NEQU 0
;  1438          1427  2                  THEN
;  1439          1428  2                      CH$MOVE( .LENGTH, KEY_2 + .I ,KEY_2 );
;  1440          1429  2
;  1441          1430  2                  EXITLOOP
;  1442          1431  2
;  1443          1432  2                  END;
;  1444          1433  2
;  1445          1434  2      ! Do rear end truncation
;  1446          1435  2      !
;  1447          1436  2      WHILE .LENGTH GTRU 1
;  1448          1437  2      DO
;  1449          1438  2
;  1450          1439  2          ! If the trailing characters are the same cut it short
;  1451          1440  2          !
;  1452          1441  2          IF .KEY_2 [ .LENGTH - 1 ] EQLU .KEY_2 [ .LENGTH - 2 ]
;  1453          1442  2          THEN
;  1454          1443  2              LENGTH = .LENGTH - 1
;  1455          1444  2          ELSE
;  1456          1445  2              EXITLOOP;
;  1457          1446  2
;  1458          1447  2      ! Set the length field
;  1459          1448  2      !
;  1460          1449  2      KEY_BUFFER_2 [ KEYR$B_LENGTH ] = .LENGTH;
;  1461          1450  2
```

```
;  1462          1451  2     RETURN
;  1463          1452  2
;  1464          1453  1     END;


                                                    KEY_1=              KEY_BUFFER_1+2
                                                    KEY_2=              KEY_BUFFER_2+2

                                        00FC   BF   BB 00000 RECOMPRESS_RECORD:
                                                             PUSHR   #^M<R2,R3,R4,R5,R6,R7>                  ;  1338
                                     5E                 04   C2 00004           SUBL2   #4, SP                          ;
                                0000'                   CF   94 00007           CLRB    KEY_BUFFER_2+1                  ;  1396
                                     56                   14   AB 9A 0000B       MOVZBL  20(KEY_DESC), LENGTH            ;  1398
                                0000'                   CF   D5 0000F           TSTL    INDEX                           ;  1402
                                                             2E 13 00013        BEQL    3$                              ;
                                     6E                      56 D0 00015        MOVL    LENGTH, (SP)                    ;  1407
                                     57                      01 CE 00018        MNEGL   #1, I                           ;
                                                             22 11 0001B        BRB     2$                              ;
                           0000'CF47          0000'CF47      91 0001D 1$:       CMPB    KEY_1[I], KEY_2[I]              ;  1412
                                                             17 13 00026        BEQL    2$                              ;
                                0000'  CF                    57 90 00028        MOVB    I, KEY_BUFFER_2+1               ;  1418
                                     56                      57 C2 0002D        SUBL2   I, LENGTH                       ;  1422
                                                             57 D5 00030        TSTL    I                               ;  1426
                                                             0F 13 00032        BEQL    3$                              ;
                     0000'  CF      0000'CF47               56 28 00034        MOVC3   LENGTH, KEY_2[I], KEY_2         ;  1428
                                                             04 11 0003D        BRB     3$                              ;  1414
                            DA              57               6E F2 0003F 2$:    AOBLSS  (SP), I, 1$                     ;  1412
                                            01               56 D1 00043 3$:    CMPL    LENGTH, #1                      ;  1436
                                                             0F 1B 00046        BLEQU   4$                              ;
                           0000'CF46          0000'CF46      91 00048           CMPB    KEY_2-1[LENGTH], KEY_2-2[LENGTH]  ;  1441
                                                             04 12 00051        BNEQ    4$                              ;
                                                             56 D7 00053        DECL    LENGTH                          ;  1443
                                                             EC 11 00055        BRB     3$                              ;
                                0000'  CF                    56 90 00057 4$:    MOVB    LENGTH, KEY_BUFFER_2            ;  1449
                                     5E                      04 C0 0005C        ADDL2   #4, SP                          ;  1453
                                        00FC   8F            BA 0005F           POPR    #^M<R2,R3,R4,R5,R6,R7>          ;
                                                             05 00063           RSB                                     ;

; Routine Size:  100 bytes,    Routine Base:  $CODE$ + 047C

;  1465          1454  1
;  1466          1455  0 END ELUDOM


;                                   PSECT SUMMARY
;
;       Name                    Bytes                           Attributes
;
;  $OWN$                         529  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;  $CODE$                       1248  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
```

```
;                               Library Statistics
;
;                                        --------- Symbols ---------      Pages       Processing
;       File                             Total     Loaded    Percent      Mapped      Time
;
;  _$255$DUA28:[SYSLIB]LIB.L32;1          18619         34          0        1000        00:01.9
;  _$255$DUA28:[CONV.SRC]CONVERT.L32;1      165         11          6          17        00:00.2




;                                       COMMAND QUALIFIERS

;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:RECLREC/OBJ=OBJ$:RECLREC MSRC$:RECLREC/UPDATE=(ENH$:RECLREC)

; Size:          1248 code + 529 data bytes
; Run Time:         00:30.1
; Elapsed Time:     01:45.3
; Lines/CPU Min:     2903
; Lexemes/CPU-Min: 15252
; Memory Used:  148 pages
; Compilation Complete
```

RECLDCL
LIS

RECLREC
LIS

RECLCTRL
LIS

RECLRMSIO
LIS

CONVMSG
LIS

CONVMAIN
LIS

CONVSORT
LIS

CONVVEC
LIS